Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Eržen

Agilen razvoj vnosnih form za potrebe shranjevanja podatkov pacientov na osnovi OpenEHR specifikacij

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

prof. dr. Miha Mraz
MENTOR

Ljubljana, 2017

Tematika naloge:

Kandidat naj v svojem delu predstavi problematiko informatizacije zdravstva. V nadaljevanju naj na osnovi vzorčne programske rešitve Think!Clinical razvije nekaj vnosnih form, potrebnih za vnos arhetipsko vezanih podatkov o obravnavanih pacientih v klinični praksi. Kandidat naj izvede vzorčno programsko analizo vnešenih podatkov na enostavnem primeru obravnave indeksa telesne mase.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela, da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih kjer so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom prof. dr. Mihe Mraza,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko in
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Janez Eržen, Ljubljana, avgust 2017.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Eržen

Agilen razvoj vnosnih form za potrebe shranjevanja podatkov pacientov na osnovi OpenEHR specifikacij

POVZETEK

V današnjem svetu se v zdravstvu še vedno uporablja kar nekaj papirnatih dokumentov, kar predstavlja veliko težav. Tudi kjer se že uporabljajo programske rešitve, so še vedno prisotne težave s souporabo teh dokumentov in podatkov na splošno. Slovenski nacionalni projekt e-Zdravje z uvedbo interoperabilne hrbtenice rešuje ravno ta problem.

V pričujočem diplomskem delu smo se dodobra seznanili s specifikacijo OpenEHR, ki je temelj arhitekturnega razvoja interoperabilne hrbtenice ter hkrati platforme *Think!Ehr Platform*^{TM}. Seznanili smo se tudi z omenjeno platformo in aplikacijo *Think!Clinical*, znotraj katere smo implementirali arhetipsko vezane vnosne forme na področju opazovanj stanja pacienta, shranjevanje podatkov vnesenih s temi formami, ter podatke uporabili za preprosto analizo pacientovega stanja telesnega razvoja na podlagi indeksa telesne mase, ki je v pomoč zdravnikom na Pediatrični kliniki ljubljanskega UKC. Pomagali smo si z orodjema "form builder" in "form renderer".

Pri tem smo ugotovili, kako hitro je mogoče vključiti preprosto vnosno formo znotraj tako obsežne zdravstvene aplikacije. Spoznali smo, kako pomembno je, da so programski produkti zasnovani čimbolj splošno, da jih je mogoče uporabiti v kombinaciji s poljubno programsko, kot tudi strojno opremo. Ta lastnost velja predvsem za pregledne API-je platforme *Think!Ehr Platform*^{TM} ter "form renderer", skupaj pa tudi za našo implementacijo vnosnih form, ki po novem delujejo tako v namizni, kot tudi v spletni aplikaciji *Think!Clinical*.

Ključne besede: e-Zdravje, OpenEHR, *Think!Clinical*, *Think!Ehr Platform*^{TM}, arhetip, predloga, vnosna forma, klinični podatki

University of Ljubljana
Faculty of Computer and Information Science

Janez Eržen

Agile development of entry forms for the needs of saving patient data using OpenEHR specification

ABSTRACT

Nowadays, a lot of documents in healthcare are still hard copies, which cause many issues during use. There are cases in which software solutions are used, but general issues with sharing those documents and data still occur. Slovenian national project e-Health with interoperable spine tries to resolve those issues.

In this thesis, one learns about OpenEHR specification, which is the base for architectural development of interoperable spine and for the *Think!Ehr Platform*^{TM}. Introduction includes the mentioned platform and application *Think!Clinical*, which implements archetypically connected entry forms regarding patient observation, saving data, which was entered with those forms, and uses the collected data for a simple analysis of the patient's state based on the body mass index, which is of great help to doctors in the Pediatric Clinic of Ljubljana University Medical Centre. For easier work, tools such as "form builder" and "form renderer" were used.

In the development process, it can be noticed how quick one can embed a simple entry form in such a large healthcare application. It has come to knowledge, how important it is to make software products as broad as possible, so they can be used in combination with different software and hardware. This feature applies mostly to clear API-s on *Think!Ehr Platform*^{TM} and "form renderer" as well as to our implementation of forms, which now works in the desktop and also web application *Think!Clinical*.

Key words: e-Health, OpenEHR, *Think!Clinical*, *Think!Ehr Platform*^{TM}, archetype, template, entry form, clinical data

ZAHVALA

Iskreno se zahvaljujem svojemu mentorju, prof. dr. Mihi Mrazu za vso pomoč pri izvedbi diplomske naloge, iskanju kvalitetnih virov ter za zanimive pogovore o problematikah informatizacije zdravstva v Sloveniji. Zahvalil bi se tudi mentorju na podjetju Marand d.o.o Anžetu Droljcu za pomoč ter nasvete pri pisanju, ter sodelavcem na podjetju Mateju Poklukarju, Igorju Horvatu, Boštjanu Lahu ter Matjažu Hiršmanu za pomoč pri implementaciji.

Poleg tega bi se na tem mestu rad zahvalil tudi svoji družini za podporo tekom študija ter med izdelavo tega diplomskega dela.

— Janez Eržen, Ljubljana, avgust 2017.

KAZALO

1 Uvod

Zdravstvo je ena od najpomembnejših dobrin ter pravic za človeka. V zadnjih dvajsetih letih prejšnjega stoletja so se zdravstvene storitve močno razvile. Zaradi potrebe po opravljanju velikih količin meritev in raziskav, je bila uvedba informatizacije v zdravstvo zelo pomemben korak naprej za razvoj zdravstvenih storitev.

V osemdesetih letih prejšnjega stoletja so se začele pojavljati prve aplikacije namenjene predvsem vodenju administracije, kadrov ter financ [1] v zdravstvu. Kasneje v devetdesetih letih se je vse več pozornosti začelo posvečati tudi shranjevanju kliničnih podatkov, ki so ključni v zdravstvu.

Tudi s strani Evropske unije je na prelomu tisočletja prihajalo veliko pobud za uvedbo enotnosti in povezovanja v zdravstvu. Zasnovali so več akcijskih programov, ki so predpisovali, v katero smer naj se razvija zdravstvo ter njegova informatizacija v državah članicah [2].

Na podlagi vizije EU ter s pomočjo njenih finančnih sredstev so se koncem prvega desetletja enaindvajsetega stoletja začeli razvijati nacionalni projekti, kot na primer tudi slovensko e-Zdravje. Slednji projekt vključuje že 17 aplikacij, ki večinoma že kažejo

pozitivne rezultate [3].

Pričujoče diplomsko delo obravnava problematiko vnašanja ter posledično tudi shranjevanja kliničnih podatkov pacientov v zdravstveni informacijski sistem. V poglavju 2 je predstavljen razvoj informacijskih sistemov v zdravstvu skozi čas, usmerjen na področje Republike Slovenije. V prvem delu je opisan razvoj nacionalnega projekta e-Zdravje. Sledi opis "interoperabilne hrbtenice", osrednjega dela informacijske infrastrukture sistema e-Zdravje (oz. eZIS), ki predstavlja osnovo za izvedbo istoimenskega projekta. V naslednjem razdelkih je opisana vloga podjetja Marand inženiring, ki je bilo izbrano za enega od glavnih realizatorjev, pri izgradnji interoperabilne hrbtenice. Kasneje se v razdelku 2.4 osredotočimo na aplikacijo *Think!Clinical* ter spoznamo z osnovnimi principi delovanja vnosnih form (razdelek 2.5).

Sledeče poglavje 3 opisuje razvoj standardov za zapis zdravstvenih podatkov od samih začetkov do današnjih dni. Kasneje se usmerimo na specifikacije OpenEHR, ki so osnova za razvoj platforme *Think!Ehr Platform*^{TM}, prav tako produkt podjetja Marand d.o.o. Na podlagi te platforme je implementirana logika za vnašanje ter shranjevanje strukturiranih zdravstvenih podatkov v praktičnem delu pričujočega diplomskega dela.

Poglavje 4 opiše glavne implementacijske cilje tega diplomskega dela, orodji "form builder" ter "form renderer" in programske jezike uporabljene pri implementaciji. Sledi opis postopka implementacije naprednih vnosnih form. V prvem delu se osredotočamo na delo s formami na splošno, v drugem pa na konkretni postopek razvoja vnosnih form za vnos "ocene stanja pacienta" ter "oceno ustne sluznice". Naslednje podpoglavje opisuje implementacijo kategorizacije pacientovega stanja debelosti na podlagi podatkov o njegovem indeksu telesne mase, ki ga izračunamo na formi za oceno stanja pacientov iz podatkov o pacientovi teži in višini.

2 Opis problema

2.1 Uvod

Področje informatizacije zdravstva je v današnjih časih v hitrem vzponu. Glede na preostale panoge, kot sta na primer bančništvo in administracija, se je informatizacija zdravstva začela odvijati dokaj pozno in je še v velikem porastu. Do tega je prišlo najbrž predvsem zaradi kompleksnosti zdravstva, ki poleg administrativnih, kadrovskih in organizacijskih problemov rešuje predvsem klinične probleme pacientov.

Zametki razvoja prvih zdravstvenih informacijskih sistemov segajo v osemdeseta leta prejšnjega stoletja [1]. V začetku so začeli z informatizacijo administracije, financ in kadrovskih zadev. Zaradi zagotavljanja kakovostnih zdravstvenih storitev se je informatizacija kasneje razširila tudi na upravljanje s kliničnimi podatki pacientov. Tovrstni klinični podatki morajo biti shranjeni v obliki, ki je prenosljiva, dostop do njih pa naj bi bil možen iz različnih naprav in lokacij.

2.2 Nacionalni projekt e-Zdravje

V Sloveniji se je v letu 2008 začelo izvajanje manjših podprojektov, ki spadajo pod nacionalni projekt e-Zdravje [3]. Začel se je na pobudo Evropske unije, ki stremi k dostopnosti in izmenjavi podatkov med državami v njej, hkrati pa od vsake države članice zahteva urejeno informatizacijo v zdravstvu. Ker je izgradnja tovrstne informacijske rešitve tehnično zelo kompleksna, je izvedba planirana postopno in v več fazah. Projekt je sofinanciran s strani EU. Konec leta 2015 je upravljanje z rešitvami razvitimi v okviru projekta e-Zdravje (2.1) prevzel Nacionalni inštitut za javno zdravje (NIJZ) [4].

Sistem e-Zdravje naj bi omogočal maksimalno interoperabilnost zdravstvenih kliničnih podatkov, kakovostnejše zdravstvene storitve in olajševal delo zdravstvenemu osebju. Integriran naj bi bil v večino zdravstvenih aplikacij, njegovi pozitivni učinki pa so že vidni v praksi.

Slika 2.1 Rešitve projekta e-Zdravje [3].

2.3 Interoperabilna hrbtenica

Osrednji del informacijske infrastrukture sistema e-Zdravja predstavlja interoperabilna hrbtenica (krajše IH) [5]. Njen glavni namen je povezati množico manjših informacijskih sistemov v enoten, centraliziran informacijski sistem. Cilji takega sistema so izboljšanje splošne dostopnosti do informacij, vpeljava boljših in nadgradnja obstoječih zdravstvenih aplikacij na višjo raven, povečanje efektivnosti celotnega zdravstvenega sistema, zmanjšanje čakalnih vrst, izboljšanje varnosti podatkov ter na podlagi zbranih informacij pomagati in voditi zdravnika do boljših odločitev. IH tako funkcionalno

omogoča izvajanje poizvedb nad zdravstvenimi dokumenti pacientov [6].

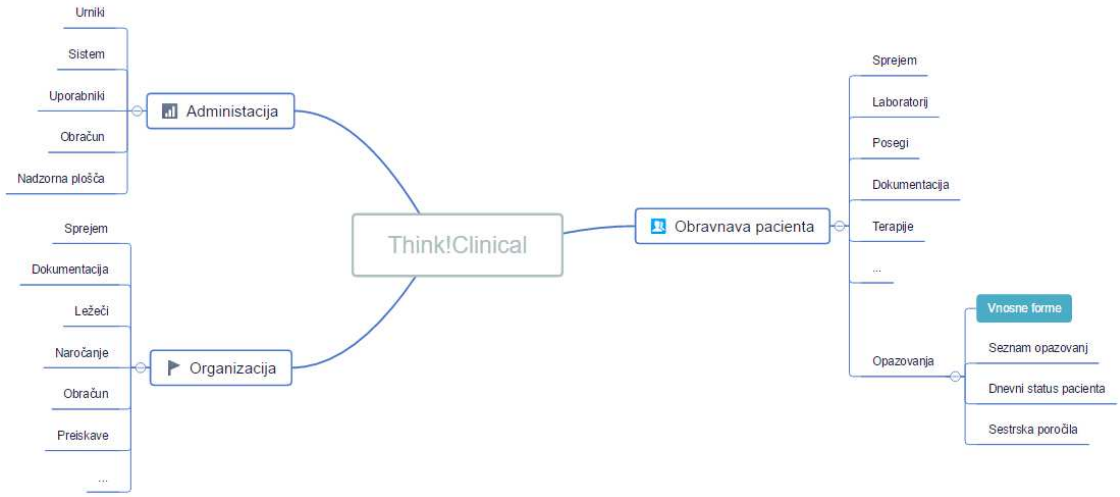
Ob vzpostavitvi sistema e-Zdravje se Ministrstvo za zdravje RS sooča s številnimi problematikami. Glede na njihove tipe tudi interoperabilnost delimo na več podtipov. Glavna sta semantična in tehnična interoperabilnost. Semantična se ukvarja predvsem z načinom predstavitve informacij. Podatki morajo biti predstavljeni v obliki, ki je razumljiva vsem uporabnikom IH, zapis pa mora biti enoten in uporabljen s strani vseh. Stremijo tudi k enotnosti na evropski ravni, a do te točke je še veliko dela.

Druga, tehnična interoperabilnost, se ukvarja s tehnično platjo problema in sicer s tem, kako povezati množico različnih IS v enoten sistem. Potrebno je definirati standarde, vmesnike in protokole, ki bodo določali način komunikacije. Težava starega sistema je namreč v tem, da v primeru, da se oseba zdravi najprej pri enem izvajalcu zdravstvenih storitev, drugi izvajalci o tem nimajo dostopnih vseh podatkov. Ker pa pri tem prihaja do številnih težav pri združevanju podatkov iz različnih virov, v veliko primerih komunikacija še vedno poteka v fizični obliki, kot npr. pri moji lastni izkušnji, ko je bilo po slikanju na zobnem rentgenu potrebno slike fizično odnesti do zobozdravnice na zgoščenki. Celotna storitev bi morala biti usmerjena v pacienta in ne v izvajalca zdravstvenih storitev, kar je ena od ključnih prednosti interoperabilnosti.

Tehnično tu obstajata dve možni rešitvi problema. Prva je uporaba infrastrukture, ki omogoča skupno hrambo gradiv, v katero vsi izvajalci zdravstvenih storitev shranjujejo tako strukturirane, kot tudi nestrukturirane (npr. zdravstvene dokumente) zdravstvene podatke. V Sloveniji se shranjujejo v centralni register podatkov o pacientih (CRPP). Zapisu v omenjenem registru pravimo “Povzetek Podatkov o Pacientu” (PPoP) [7]. Drugi možni način izmenjave informacij med dvema zdravstvenima ustanovama je direktna “Point-to-Point” komunikacija, vzpostavljena preko VPN (angl. *Virtual Private Network*).

2.4 Vloga podjetja Marand v informatizaciji slovenskega zdravstva

Pomemben prispevek k izgradnji interoperabilne hrbtenice je prispevalo podjetje Marand d.o.o. s svojo platformo *Think!Ehr Platform*^{TM}, ki skrbi za trajno in strukturirano shranjevanje zdravstvenih podatkov. Platforma temelji na OpenEHR specifikaciji, ki je napisana v skladu z načeli o semantični interoperabilnosti in zato omogoča prenosljivost



Slika 2.2 Vsebinska shema aplikacije *Think!Clinical* se deli na tri glavne dele in sicer na administrativni del, organizacijski del in obravnavo pacientov.

podatkov med različnimi aplikacijami, ustanovami ali celo med državami.

Eden od produktov omenjenega podjetja, ki temelji na tej platformi, je klinični informacijski sistem *Think!Clinical*. Trenutno se uporablja na Pediatrični kliniki Univerzitetnega kliničnega centra Ljubljana in na Onkološkem inštitutu v Ljubljani. Sistem je sestavljen iz zalednega dela, ki povezuje različne zunanje storitve in skrbi za upravljanje s podatki ter iz uporabniške aplikacije. Aplikacija uporabnike vodi čez proces dela s pacienti, administracijo, obračun, omogoča pregled stanja pacientov, vnašanje meritev in vitalnih znakov, predpisovanje zdravil itd. *Think!Clinical* je le ena od mnogih aplikacij, ki svoje podatke shranjujejo preko te platforme.

Aplikacija se deli na tri glavne dele. Prvi je administrativni del, ki skrbi za urejanje urnikov zdravnikov, uporabnike in pravice, obračun opravljenih storitev in nadzor delovanja aplikacije ter sistemske nastavitve. Drugi del obravnava delo z zdravstvenimi procesi na ravni organizacij in oddelkov, ter omogoča pregled pacientov in njihovo nadaljnjo obravnavo. Tretji del skrbi za pacienta samega in njegov celotni proces zdravljenja, od sprejema, ambulantne obravnave, obiska laboratorija, predpisovanja terapij in zdravil, izpisa dokumentacije, pa do samih kliničnih meritev in opazovanj vitalnih znakov. Prav na ta del aplikacije se bomo osredotočili v pričujočem diplomskem delu. Grafična ponazoritev sheme aplikacije je predstavljena na sliki 2.2.

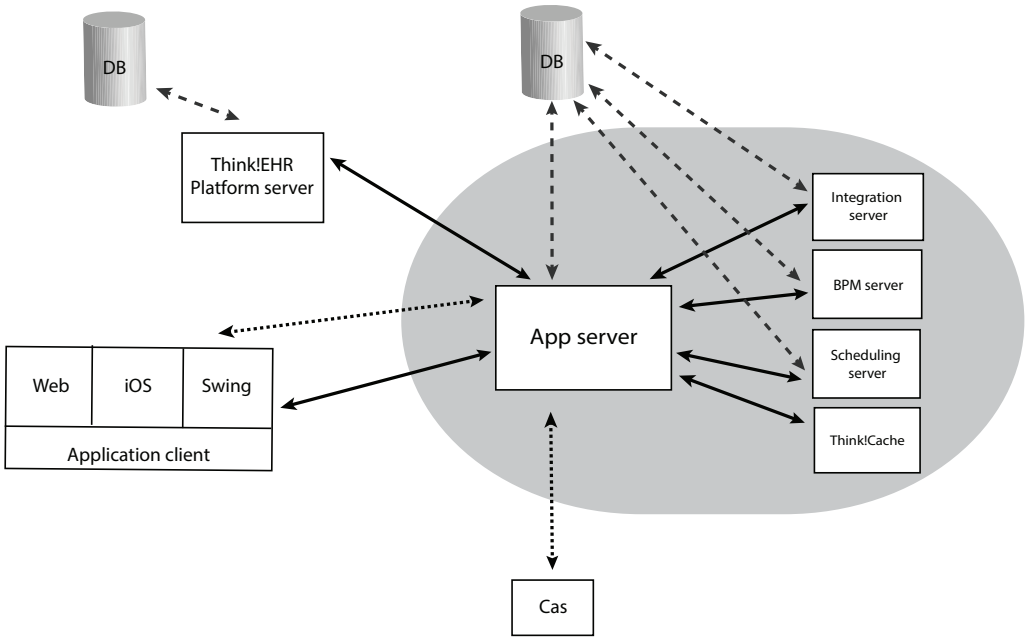
Kot že omenjeno aplikacija *Think!Clinical* za shranjevanje kliničnih podatkov uporablja platformo *Think!Ehr Platform*^{TM}, kar je glavni razlog za interoperabilnost sistema. Aplikacija namreč za shranjevanje kliničnih podatkov uporablja ločen strežnik platforme, ki je dostopen tudi drugim aplikacijam. Do njega lahko dostopa direktno preko RPC-ja (angl. *Remote Procedure Call*) ali z uporabo API-ja, ki ga platforma ponuja, torej po protokolu HTTP.

S fizičnega vidika je aplikacija sestavljena iz več manjših strežnikov, od katerih vsak opravlja točno določeno nalogo (npr. integracijo zunanjih aplikacij, upravljanje s poslovnimi procesi in urniki, predpomnjenje podatkov), vsi pa komunicirajo in nudijo podatke glavnemu aplikacijskemu strežniku. Glavni strežnik je hkrati tudi posrednik med aplikacijo in platformo. Večina klicev na slednjo gre namreč preko njega. *Think!Clinical* uporablja eno instanco "Oracle" podatkovne baze, v kateri so narejene različne sheme za različne sklope podatkov. Vanjo uporabniške aplikacije shranjujejo administrativne in kontekstne podatke za aplikacijo. V Oracle podatkovnih bazah shema predstavlja uporabniški račun ter zbirko tabel, pravic in ostalih metapodatkov. Ni pa nujno, da shema vsebuje vse tabele v podatkovni bazi. *Think!Ehr Platform*^{TM} uporablja popolnoma ločeno podatkovno bazo.

Uporabniška aplikacija je dostopna kot namizna, spletna ali mobilna (iOS) aplikacija. Na aplikacijski strežnik se povezuje direktno preko RPC-ja (namizna aplikacija) ter preko HTTP zahtevkov. Več podrobnosti arhitekture aplikacije je predstavljenih na sliki 2.3.

2.5 Vnosne forme kot osnova za shranjevanje podatkov

Klinične podatke v aplikacije, pa najsi bodo spletne ali namizne, običajno vnašamo preko vnosnih form. Pri običajnem razvoju aplikacij analitik najprej določi, kateri podatki so zahtevani na določeni vnosni maski v aplikaciji, razvijalec pa se nato loti izdelave uporabniškega vmesnika. Na neko risalno površino umesti vnosna polja in njihove oznake. Forma se validira ob spremembi vrednosti v polju ali pa ob končni oddaji forme, ko uporabnik pritisne gumb za shranjevanje. Ob shranitvi forme in uspešni validaciji je potrebno podatke preko neke storitve (lahko gre za REST API, ali direkten klic) shraniti v podatkovno bazo. Še pred tem jih je potrebno s polj na formi prevesti v ustrezno obliko. Če se pojavi zahteva po novem polju na formi, mora razvijalec dodati novo polje, popraviti validacijo ter shranjevanje.



Slika 2.3 Fizična shema rešitve sistema *Think!Clinical*. *Cas strežnik* služi za varno komunikacijo z zunanjimi sistemi, ter ponekod tudi znotraj aplikacije.

Tudi podjetje Marand je imelo večino form narejenih po tem principu. V zadnjih letih pa se je začela prenova na novo, moderno, lahko spletno tehnologijo, saj to narekujejo smernice razvoja aplikacij. Poleg tega so dandanes vedno večje potrebe po dostopu do aplikacij od koderkoli. Podjetje Marand se zato poleg razvoja namizne aplikacije nagiba k temu, da bi večji del funkcionalnosti aplikacije *Think!Clinical* prenesli tudi v njihovo spletno aplikacijo *Think!Clinical Web*.

Praktični del pričujoče diplomske naloge obsega predelavo večine modula opazovanj znotraj aplikacije *Think!Clinical* v moderne spletne tehnologije. Modul vsebuje večinoma vnosne maske namenjene spremljanju vitalnih znakov pacientov, sprememb na koži, v sklepih, ocenjevanju bolečin, ter še nekaterim preostalim ocenjevalnim lestvicam.

Glavna orodja pri razvoju vnosnih form so načrtovalec vsebine form (angl. *form builder*), "form renderer", ki generira "AngularJS" vnosne forme glede na opis zgrajen z "builderjem" ter *Think!Ehr Platform*^{TM}, ki s svojimi API-ji omogoča preprosto shranjevanje podatkov.

2.5.1 Postopek razvoja vnosne forme

Postopek razvoja vnosne forme poteka v sledečih korakih:

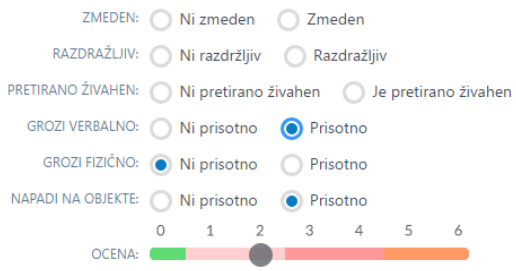
1. S spletnim urejevalnikom (angl. *form builder*) zgradimo vnosno formo:
   (a) v "form builder"-ju se izbere ustrezno predlogo, ki predstavlja nabor vseh možnih polj;
   (b) izbrana polja prenesemo na formo po principu "drag&drop";
   (c) po potrebi nastavimo dodatno validacijo na formi, polja stilsko oblikujemo in dodamo zunanje terminologije, če je to potrebno;
   (d) formo shranimo (shrani se v JSON obliki);
   (e) formo naložimo na ustrezni strežnik;
2. V aplikaciji na akcijo uporabnika (npr. klik) odpremo vnosno formo:
   (a) pokličemo ustrezen REST klic, ki pridobi JSON opis forme;
   (b) ta opis podamo "form renderer"-ju, ki glede na opis v podani HTML element izriše vnosno formo;
   (c) "renderer" poskrbi tudi za ustrezno validacijo forme;
3. Formo, ki ne vsebuje kompleksnejših podatkov shranimo tako, da najprej na formi s klicem ustrezne funkcije pridobimo v JSON zapisane podatke, nato pa jih shranimo s POST klicem na REST API.

2.5.2 Glavni izzivi pri razvoju vnosnih form

Opis v prejšnjem razdelku deluje za najbolj preproste forme, seveda pa v praksi vse skupaj ni tako preprosto. Sledi opis nekaterih realnih izzivov, ki se pojavljajo ob implementaciji form:

- V realnosti se pogosto srečamo s povezavami med različnimi atributi in vrednostmi. Preprost primer je npr. soodvisnost vrednosti indeksa telesne mase od teže in višine. V primeru, da se spremeni ena od teh treh vrednosti, se morata preračunati ostali dve. V praksi smo to reševali z uporabo "JavaScript" funkcij.

- Pri izdelavi več povezanih form lahko prihaja do soodvisnosti med polji dveh različnih form, zato je treba vrednosti z ustreznimi funkcijami, ki se kličejo na ustrezen dogodek, prenašati in združevati.
- Večkrat se stvar zaplete pri validaciji, saj lahko prisotnost ene vrednosti zahteva, da je prisotna tudi neka druga. Take probleme smo reševali pred končnim REST klicem.
- Ker aplikacija omogoča spremljanje kliničnega stanja pacienta v realnem času, je potrebno v nekaterih primerih podatke pridobiti direktno iz medicinskih naprav, jih vnesti v formo in shraniti v sistem.
- Pogosto za izračun neke vrednosti na formi oz. le za prikaz potrebujemo pacientove klinične ali pa osebne podatke, ki jih pred inicializacijo pridobimo s klicem na REST, ki vrača podatke iz podatkovne baze in jih nato uporabimo na formi.
- Ker sta videz vmesnikov in uporabniška izkušnja zelo pomembna, je v nekaterih primerih za boljšo predstavo uporabnika nekatere izračunane vrednosti ali polja za izbiro smiselno prikazati z grafično obogatenimi komponentami. To so razni animirani drsniki, "imagemap"-i in podobno. Tudi ta scenarij je pokrit s funkcionalnostmi "form renderer"-ja. Vsak element na formi lahko poljubno oblikujemo s kaskadnimi slogi, mu dodajamo nove elemente, ali pa ga zamenjamo s povsem drugo komponento, njegov "logični" model pa s tem ohranimo. Primer take komponente je predstavljen na sliki 2.4.

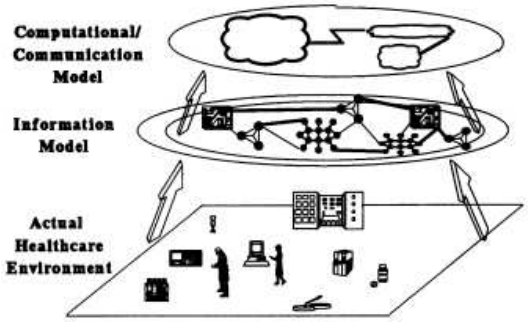


Slika 2.4 Obogatena grafična komponenta prikazuje oceno stanja agresivnosti pacienta glede na vnesene vrednosti, v osnovi pa je ta ocena le navadno številsko polje.

3 Standardi za zapis zdravstvenih podatkov

3.1 Zgodovina

Koncem devetdesetih let je organizacija MEDIX, ustanovljena s strani Toma Rutt-a, strokovnjaka za standarde, razvila prvi koncept podatkovnega modela v zdravstvu za splošno rabo [8]. Ideja je nastala na konferenci MEDINFO leta 1986, ko so mednarodni strokovnjaki, raziskovalci in razvijalci sistemov prišli do spoznanja, da je zaradi heterogenosti v delovanju zdravstvenih sistemov potrebna sistemizacija in načrtovana informatizacija. Svetovno znana organizacija IEEE, ki se ukvarja z izobraževanjem, standardizacijo in tehničnim napredkom na področju tehnologije, je še istega leta ustanovila komite MEDIX. Že v prvi fazi so predvideli, da mora biti standard zasnovan na mednarodni ravni in pokrivati različne dejavnike odvisne od okolja in sicer različne tipe, protokole, strukturo podatkov ter fizična omrežja. Začeli so s pripravo konceptualne sheme, ki je kasneje prerasla v informacijski model. Vsakovrstna komunikacija naj bi potekala po predpisanih protokolih z upoštevanjem te informacijske sheme.



Slika 3.1 Infrastrukturni model organizacije MEDIX [8].

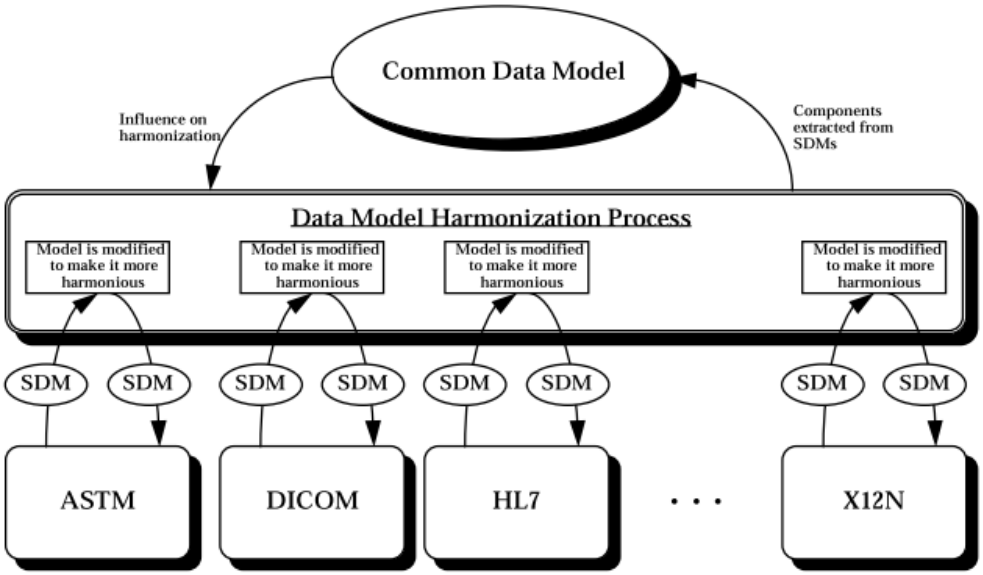
Leta 1988 je bila v Evropski uniji vzpostavljena iniciativa za napredno informatiko v medicini (Advanced Informatics in Medicine - AIM). Njena naloga je bila okrepiti in povezati skupnost, ter uvesti znanstveno in tehnično enotnost. Na podlagi te iniciative je leta 1992 nastal projekt GEHR (The Good European Health Record) [9]. Njegovi člani so s svojimi izkušnjami ter predanostjo bili boj z različnimi izzivi in z dolgotrajnim prototipiranjem in testiranjem konceptov počasi gradili objektno naravnano EHR arhitekturo, ki je temeljila predvsem na izkušnjah in problemih zdravstva članic EU.

Kot rezultat hitrega razvoja podjetij, ki so razvijala svoje standarde za interoperabilnost v zdravstvu, je bilo ustanovljenih več organizacij, ki so skrbele za komunikacijo in sinhronizacijo teh podjetij med seboj.

MEDIX je že takoj na začetku uvedel objektno usmerjen referenčni model, kar je za takratne čase pomenilo velik korak naprej. Leta 1993 je Sigurd From s svojo razvojno ekipo razvil projekt CEN, ki je bil glavni predhodnik standarda HL7. Člani njegove ekipe so v sodelovanju z udeleženci GEHR sestavili predlog standarda CEN (ENV 12265), ki je s tem objavil jasno predstavljene rezultate raziskav ekipe GEHR. Leta 1994 se je projekt GEHR končal, z naslednjim letom pa je njegov nadaljni razvoj prevzel CHIME (Centre for Health Informatics and Multi-Professional Education) ustanovljen s strani UCL (University College London). Razvoj se je zaradi nenaklonjenosti Evrope projektu GEHR v naslednjih letih preselil v Avstralijo.

IEEE je oblikovala združenje "Joint Working Group for Common Data Model" oz. JWG-CDM, ki je v letih od 1992-96 združevalo glavne akterje v razvoju zdravstva in se na koncu odločilo, da za glavno organizacijo v razvoju skupnega zdravstvenega modela izbere organizacijo HL7, ki je takrat kazala največ interesa za razvoj [10]. Omenjeno

združenje je razvilo CDM (angl. *Common Data Model*), ki je opisoval dobre prakse in priporočljive procedure za razvoj komponent za izmenjavo zdravstvenih podatkov. S procesom harmonizacije naj bi poskrbel za podatke (njihovo strukturo) iz podsistemov, da bi se prenesli v skupni podatkovni model predstavljen na sliki 3.2.



Slika 3.2 Infrastrukturni model organizacije JWG-CDM [8].

Sestanki združenja JWG-CDM so bili pogosto v gosteh pri HL7, saj so številni razvijalci HL7 podpirali koncepte standarda CDM, katerega meta-model je predstavljen na sliki 3.2. Leta 1996 se je, ko je bil narejen osnutek standarda HL7 verzije 3, začel razvoj standarda HL7 RIM. Glavni pobudnik in začetnik je bil Abdul-Malik Shakir. Slednji standard je uporabljal obstoječe modele iz različnih virov, tako iz starejših standardov HL7, kot tudi MEDIX in ostalih podatkovnih modelov. Vse te modele je s postopkom harmonizacije združil v enotno obliko.

Leta 1998 v ospredje kot poenostavitev modela RIM stopi standard USAM (Unified service Action Model), ki je zaradi velikega števila razredov in atributov v modelu stremel k generalizaciji, združevanju ter abstrakciji razredov, čez dve leti pa je sledila še verzija 2. Razredi so postali bolj splošni in se jih je specializiralo z uporabo različnih terminologij, njihovo število pa je tako drastično upadlo. Velikokrat je obravnavan le kot razširitev standarda RIM. Združeni standard je navdušil veliko število prostovoljcev za HL7, da so poustvarili obstoječe modele in specifikacije. Julija 2003 je bil priznan

s strani organizacije ANSI, kasneje med leti 2005 in 2007 pa še kot ISO standard [11]. Standard RIM verzije 2 trenutno predstavlja osnovo za vse načine modeliranja z HL7.

3.2 ISO13606

Standard ISO13606 oz. že prej imenovani CEN je normativ na področju Evrope. Njegov glavni namen je doseči interoperabilnost v komunikaciji z EZZ (elektronskimi zdravstvenimi zapisi) [12]. Definiral naj bi način komunikacije z EZZ med informacijskimi sistemi ter podatkovnimi repozitoriji. Standard ločuje informacijsko plat in znanje, zato arhitekturi narejeni po njegovih predpisih pravimo, da je dualna. Informacijsko plat določajo entitete vsebovane v referenčnem modelu, ki hranijo informacije iz EZZ. Semantiko in podatke nosijo arhetipi, ki znanje formalno definirajo in omejijo nabor možnih vrednosti. Primer arhetipa bi bil lahko na primer "družinska zgodovina" ali "meritev glukoze". Arhetipi omogočajo trajno shranjevanje in odpornost podatkov na spremembe. Standard je bil kasneje uporabljen kot glavna osnova za OpenEHR. Verzija ISO 13606-2 še vedno predstavlja specifikacijo za arhetipe v OpenEHR.

3.3 HL7

Standarde nastale pod okriljem globalne organizacije Health Level Seven International, katerih razvoj je opisan v razdelku 3.1, razvrščamo v več različnih kategorij. Za primerjavo s standardom OpenEHR bo dovolj, če na tem mestu omenim le primarne [13]. Za standardizacijo transakcij in sporočanje v zdravstvenih informacijskih sistemih se uporabljata dve glavni verziji (2.x in 3) standardov HL7. Primer takega zapisa (verzije 2), meritev ravni glukoze v krvi, je predstavljen v zapisu 7.

```
MSH|^~\&|CERNER||PriorityHealth||||ORU^R01|Q479004375T431430612|P|2.3|
PID|||001677980||SMITH^CURTIS||19680219|M|||||||||929645156318|123456789|
PD1||||1234567890^LAST^FIRST^M^^^^^NPI|
OBR|1|341856649^HNAM_ORDERID|00000200632600236 2|648088^Basic Metabolic Panel|||20061122151600|||||||||
        1620^Hooker^Robert^L||||||20061122154733|||F||||||||||20061122140000|
OBX|1|NM|GLU^Glucose Lvl|59|mg/dL|65-99^65^99|L|||F|||20061122154733|
```

Zapis 3.1 Primer HL7 zapisa po standardu verzije 2

HL7 s CDA (angl. *Clinical Document Architecture*) predpisuje model kliničnih dokumentov, ki temeljijo na standardu za sporočanje verzije 3. Specificira kodiranje, strukturo ter tudi semantiko kliničnih dokumentov za izmenjavo [14]. ZDA so CDA razširile s standardom CCD (angl. *Continuity of Care Document*) za izmenjavo poročil. SPL (angl.

Structured Product Labeling) definira predpisano obliko informacij, s katerimi so opremljena zdravila. Eden od standardov definira celo vizualno podobo HL7 vmesnikov za shranjevanje medicinskih podatkov v aplikacijah za končne uporabnike. Vsi omenjeni standardi so opisani v notaciji XML, ki je bila v devetdesetih letih prejšnjega stoletja zelo popularna na vseh področjih informatizacije in je še danes.

3.4 OpenEHR

Odprta skupnost OpenEHR skrbi za interoperabilnost pri prenosu zdravstvenih podatkov iz stvarnega sveta v elektronsko obliko. Nastala je na podlagi zahteve Davida Ingrama, člana CHIME-ja, po ustanovitvi fundacije, ki bi bila usmerjena v upravljanje s kliničnimi podatki. Temelji na že predhodno definiranih standardih ISO-13606 (CEN) ter GEHR.

Glavna prednost OpenEHR je strukturiran zapis zdravstvenih podatkov [15]. Omogoča klinično podporo odločanju (angl. *decision support*), izboljšuje varnost pacientov, omogoča zapis podatkov v registre, zdravstveni pregled nad populacijami ljudi, poslovno inteligenco, zdravstvene raziskave, personalizacijo zdravstva itd. S svojim poizvedovalnim jezikom AQL (angl. *Archetype Query Language*) in strukturiranim zapisom podatkov, kjer je vsak podatek dostopen na točno določenem mestu, preko določene poti (angl. *path*) dopušča brezmejne možnosti za razvijalce.

3.4.1 Osnovni koncepti

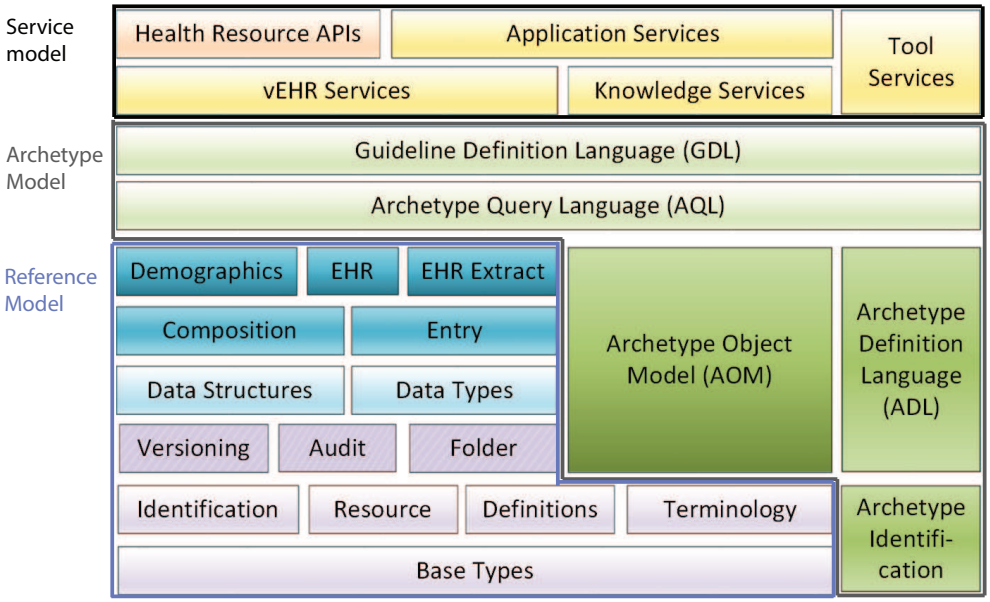
OpenEHR pristop temelji na storitveno orientirani arhitekturi, katere klinični modeli podatkov so načrtovani s strani strokovnjakov z medicinskega področja v sodelovanju z informatiki. Modeli naj bi bili načrtovani na svetovni ravni in sprejeti s strani večine zdravstvenih organizacij. Principi OpenEHR so definirani v zbirki specifikacij, narejenih s strani istoimenske fundacije.

Kot vsi ostali principi za razvoj zdravstvenih informacijskih sistemov tudi OpenEHR temelji na EHR. EHR (angl. *Electronic Health Record*) je elektronski zdravstveni zapis posameznega pacienta, ki vključuje različne klinične podatke in karakteristike pacienta. V slovenskem prostoru uporabljamo kratico EZZ (elektronski zdravstveni zapis). Predstavlja nekakšen osnovni okvir, znotraj katerega so shranjeni vsi podatki obravnav posameznega pacienta.

V grobem delimo OpenEHR specifikacijo na tri glavne sklope (slika 3.3):

- servisni model (angl. *Service Model - SM*);

- arhetipski model (angl. *Archetype Model - AM*);
- referenčni model (angl. *Reference Model - RM*);

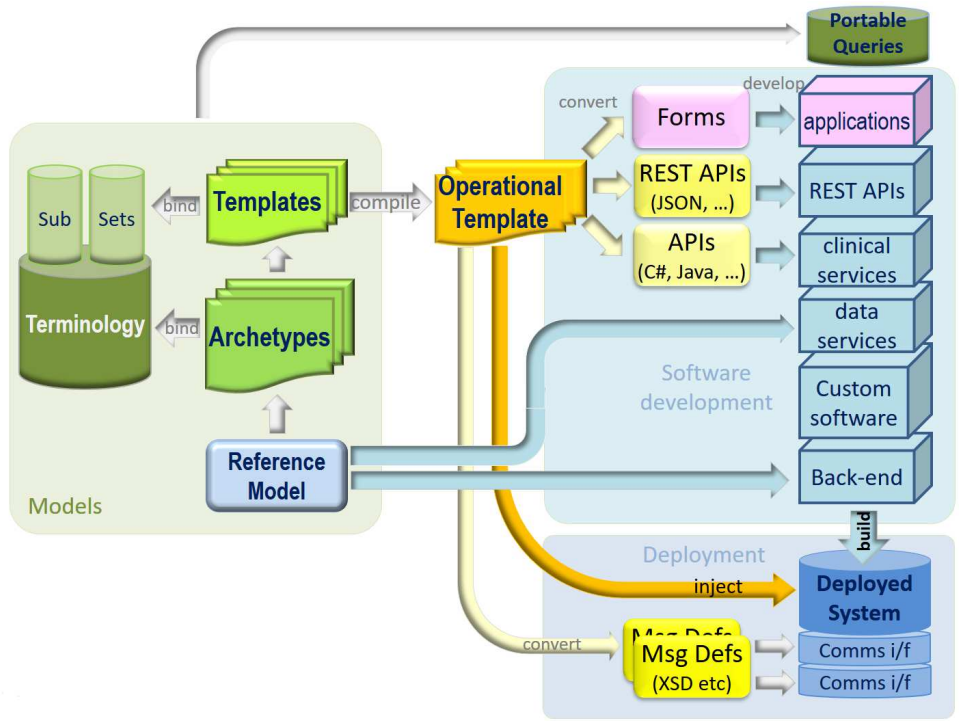


Slika 3.3 Globalna OpenEHR paketna struktura [16].

V praksi se srečujemo s petimi glavnimi deli principa OpenEHR, ki so sledeči:

- referenčni model;
- arhetipi;
- template-i;
- poizvedovalni jezik AQL [17];
- API fizične platforme;

Predhodno navedeni principi so predstavljeni na sliki 3.4. Prvi štirje, prikazani na levi strani slike (na sliki manjka AQL) predstavljajo koncepte, ki so del specifikacije, desna stran slike pa je stvar implementacije posameznega informacijskega sistema. S takim konceptualnim pristopom specifikacija OpenEHR omogoča implementacijo v poljubnem programskem jeziku, z različnimi načini izpostavljanja storitev (REST, SOAP...), neodvisno od strojne opreme in operacijskega sistema.



Slika 3.4 Konceptualni model OpenEHR [18].

3.4.2 Referenčni model

Referenčni model določa logično strukturo za shranjevanje v obliki EZZ in demografskih podatkov (generalije). Pojem demografski podatki v okviru OpenEHR predstavlja vse pacientove osebne podatke - generalije. Omenjeni model predstavlja napotke in dobre prakse, kako naj bi bile stvari narejene, ne pa dejanske sheme modela uporabljenega v praksi. Definira osnovne podatkovne tipe, njihove omejitve, osnovne podatkovne strukture, ki se jih nato uporablja za gradnjo arhetipov, določa proces upravljanja z verzijami, identifikacijo, dostop do podatkovnih virov itd.

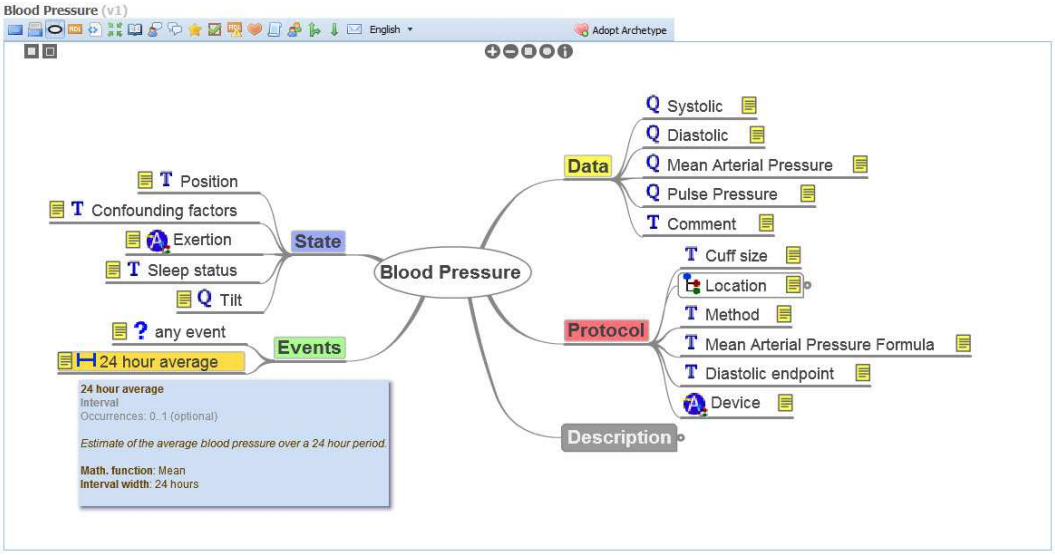
Osnovne podatkovne tipe, ki so v končnih aplikacijah predstavljeni na vnosnih formah, delimo na štiri glavne skupine. V razlagi podatkovnega tipa je pripisan tudi praktični prikaz na vnosni formi, ki seveda ne spada pod referenčni model in je stvar same implementacije. Glavne skupine podatkovnih tipov so sledeče:

- tekstovni podatki:
  - DV_TEXT - prosti tekst, predstavljen s tekstovnim poljem;
  - DV_CODED_TEXT - nabor možnih vrednosti; vsaka ima svojo unikatno kodo in tekstovno vrednost; tip je običajno predstavljen s spustnim menijem; pogosto je prisotna tudi dodatna opcija "drugo", ki poleg izbire vrednosti iz nabora omogoča vnos prostega teksta;
- DV_BOOLEAN - omogoča izbor dveh vrednosti in sicer drži ali ne drži; v praksi se prikazuje s potrditvenim poljem (angl. *checkbox*), v nekaterih primerih pa potrebujemo tudi nedefinirano vrednost (angl. *null*), zato posledično polje predstavimo s tremi "radio button"-i;
- časovni podatki:
  - DV_DATE - vnos datuma predstavljen z izbiralcem datuma (angl. *datepicker*);
  - DV_TIME - vnos časa;
  - DV_DATETIME - vnos kombinacije časa in datuma;
- količinski podatki:
  - DV_COUNT - vnos številnosti, npr. število odmerkov določenega zdravila na časovno enoto; predstavljeni so s številskim poljem;
  - DV_PROPORTION - vnos deleža, npr. zasičenosti izdihanega zraka s kisikom; predstavljeni so z dvema poljema, kjer eno predstavlja delež, drugo pa celoto oz. količnik; če gledamo delež v procentih, se drugo polje običajno izpusti;
  - DV_QUANTITY - vnos količine določene snovi s pripadajočo enoto; predstavljeni so z dvema poljema in sicer s številskim za vnos količine ter dodatnim poljem DV_CODED_TEXT, ki uporabniku omogoča izbiro enote, s katero so bili izmerjeni vneseni podatki.
  - DV_DURATION - vnos trajanja nečesa, npr. trajne infuzije; predstavimo jih z več številskimi polji ali preračunamo v datume;

Poleg omenjenega referenčni model definira tudi osnovne kontejnerje za shranjevanje podatkov. Določa strukturo osnovnega EZZ zapisa, demografskih podatkov, podatkovnega zapisa (angl. *composition*) ter ostalih podatkovnih struktur.

3.4.3 Arhetipi

Kot je že omenjeno v predhodnem razdelku, referenčni model določa le definicijo strukture podatkov. OpenEHR temelji na pristopu dvonivojskega modeliranja, torej ločuje jezikovna in vsebinska pravila. Nad referenčnim modelom je zato definiran arhetipski model ali množica arhetipov. Slednji predstavljajo najširšo možno definicijo podatkov, ki so zanimivi za klinično analizo posameznih primerov uporabe. Uporabljeni naj bi bili na globalni ravni, dostopni pa so na CKM (angl. *Clinical Knowlede Manager*) strežnikih, tako globalnem, kot tudi na nacionalnih. Načrtovani so s strani mednarodnih kliničnih strokovnjakov, ki imajo tudi tehnično znanje o EZZ sistemih. Definicija arhetipov je omejena s specifikacijo jezika ADL, v katerem so narejeni. Fundacija hkrati ponuja tudi načrtovalska orodja za razvoj arhetipov. Na sliki 3.5 je prikazan in opisan primer arhetipa.



Slika 3.5 Primer arhetipa za krvni pritisk. Arhetip je sestavljen iz več tipov podatkov. Podatkovna sekcija (angl. *data*) vsebuje izmerljive podatke, ki jih vnese zdravnik. Protokol opisuje na kakšen način, po kateri metodi in s katero napravo smo podatek pridobili. Stanje (angl. *state*) opisuje stanje pacienta ob opazovanju. Dogodki (angl. *events*) opisujejo neko ponavljajoče se dogajanje/meritev, vezano na čas. Prisotna je tudi sekcija za opis (angl. *description*), ki omogoča vnos prostega teksta [19].

3.4.4 Predloge

Naslednji višji nivo nad arhetipi so predloge (angl. *template*). Predloga je nabor podatkov iz različnih arhetipov, pripravljen za posamezni primer uporabe. To je lahko vnosna forma, sporočilo, dokument ali pa skupina med sabo povezanih podatkovnih skupin. Običajno so izdelana na podjetju, ki izdeluje svoj produkt, ponekod pa uporabljajo tudi skupne nacionalne predloge za določene scenarije. V predlogah načrtovalec določi, katere podatke iz arhetipov mora predloga vsebovati in jim na ravni predloge tudi omeji številnost, dovoljene meje, nabor vrednosti, določimo pa lahko tudi zunanje terminologije itd.

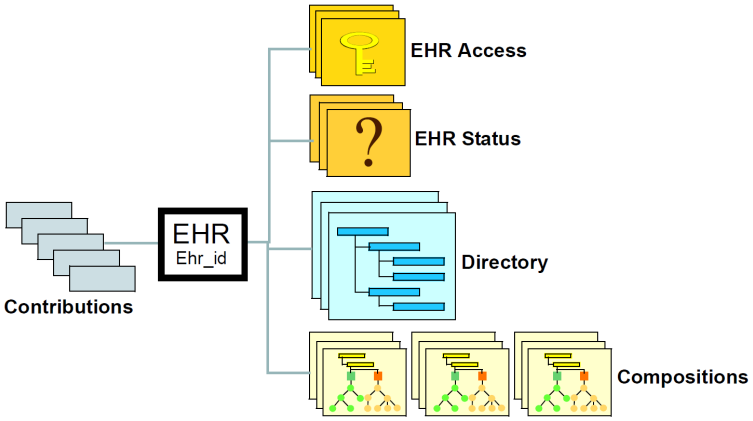
Predloge so običajno sestavljene s strani analitika in zdravstvenega osebja. Njihov razvoj poteka v več iteracijah. Zdravniki pogosto niso povsem vešči OpenEHR sistemov in v posameznih primerih uporabe jih včasih analitik z informiranjem o tem, kateri vsi podatki so možni za določeno področje, opomni na pomembne koncepte, ki bi bili mogoče tudi uporabni za raziskave. Proces razvoja poteka tako, da analitik za primer uporabe v predlogo vključi ustrezne arhetipe, glede na to, katere informacije ga zanimajo. Nato se z zdravstvenim osebjem dogovori, kateri podatki so zanj sploh potrebni in nepotrebne izloči iz predloge. Nato sledi določitev dodatnih omejitev ter po potrebi dodajanje zunanjih terminologij za specifične probleme. Tak proces običajno traja nekaj iteracij, da pride do končnega konsenza med zdravnikom in analitikom.

Predloga je tekom razvoja zapisana v datoteki tipa ".oet" v XML obliki. Običajno se take datoteke ureja z grafičnimi urejevalniki. Omenjena datoteka vsebuje povezave na različne arhetipe (zunanje datoteke) in omejitve nad njimi ter nekatere druge metapodatke. Preden bo predloga pripravljena za uporabo v produkcijskem okolju, jo je treba pretovoriti v **operativno predlogo** (angl. *operational template* - OPT). Slednja je samostojna in vsebuje vse podatke o arhetipih, podatkovnih tipih, terminologije s prevodi, torej vse kar predloga potrebuje za samostojno delovanje. Kot primer, predloga "Diabetes mesečni pregled" v razvojni verziji vsebuje okoli 200 vrstic dolgi XML dokument, ob pretvorbi v operativno predlogo pa ta velikost naraste na okoli 25.000 vrstic.

3.4.5 EZZ, EMR in PHR

Kot že omenjeno se vsi klinični zapisi za posameznega pacienta vpisujejo v njegov EZZ. Slednji poleg shranjevanja kliničnih podatkov podpira tudi shranjevanje podatkov, ki se tičejo plačevanja, upravljanja kakovosti storitev, planiranja virov ter nadzora in statistike,

torej na kratko administrativnih podatkov. Namen EZZ je namreč omogočiti celovito zdravstveno oskrbo pacienta [20].



Slika 3.6 Struktura EZZ zapisa [21].

EZZ zapis pacienta naj ne bi vseboval nobenih demografskih podatkov (generalij) [22]. Eden od osnovnih principov OpenEHR je ločitev demografskih od ostalih podatkov, kar naj bi zagotavljalo težjo sledljivost in manjšo možnost kraje podatkov. OpenEHR predpostavlja uporabo fizično ločenih demografskih repozitorijev in tudi predpisuje strukturo demografskih podatkov z arhetipi.

V praksi so demografski podatki pogosto shranjeni v ločenih bazah. Za povezavo EHR podatkov z zunanjimi sistemi se uporablja enotni identifikator Ehr_id. Osnovni zapis lahko poleg enoličnega identifikatorja vsebuje tudi identifikator zunanjega sistema, ki je zapis ustvaril, ter časovni žig, ko je bil zapis narejen. Teh treh podatkov po vnosu ni več mogoče spreminjati. Kot je vidno iz slike 3.6, EZZ zapis vsebuje podatke o:

- EZZ dostopu (angl. *access*), ki definira pravice in nastavitve dostopa do zapisa (torej privzeti način ter dovoljenja posameznim uporabnikom);
- EZZ statusu, ki vsebuje trenutno stanje in kontrolne informacije o EZZ med katerimi je tudi eksterni identifikator pacienta, ki je povezan s trenutnim EZZ, lastništvo zapisa, zadnja posodobitev itd.;
- strukturi map (angl. *directory*), ki opisuje hierarhično strukturo organizacije zapisov pacienta po mapah, s tem, da se lahko en zapis za razliko od običajnih direktorijskih struktur pojavlja v več mapah naenkrat;

- podatkovnih zapisih (angl. *composition*), ki vsebujejo vse klinične in administrativne vnose podatkov za pacienta; tudi za administrativne vnose so definirani specializirani arhetipi;

- prispevkih (angl. *contribution*), ki beležijo vse spremembe v pacientovem EZZ; vse predhodno naštete strukture so namreč verzionirane; noben zapis se torej ne posodobi direktno, ampak se vedno naredi nova verzija obstoječega zapisa, da noben podatek ne mora biti izgubljen;

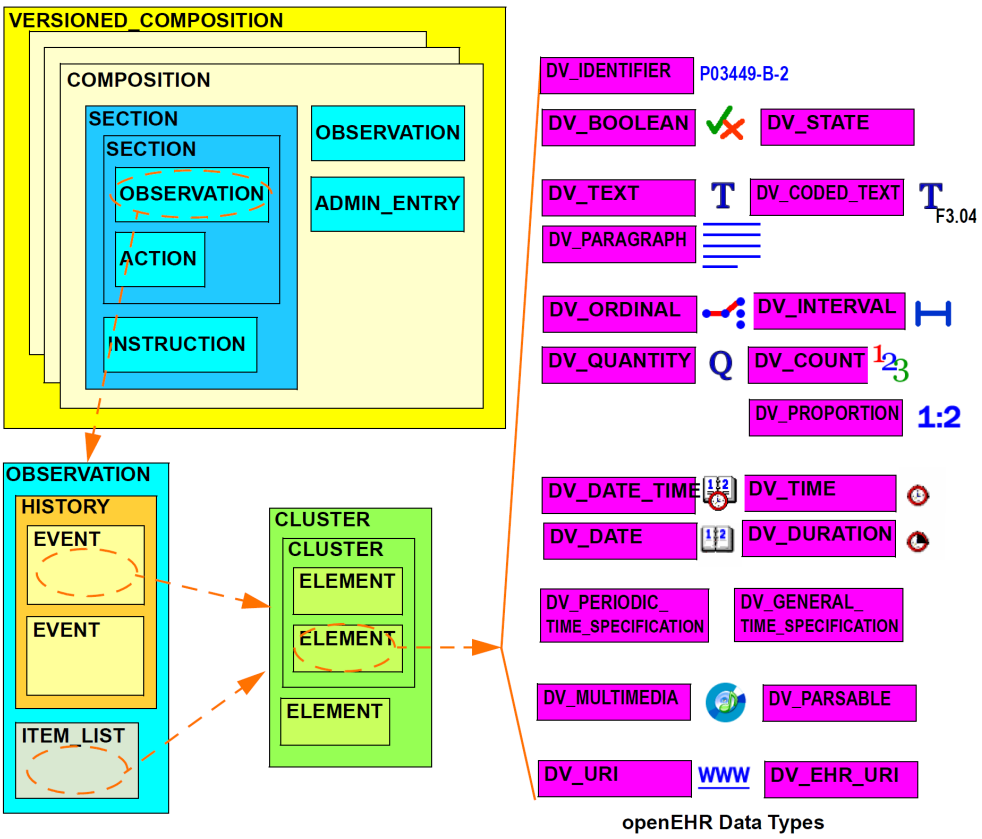
V zdravstvu se pogosto uporablja tudi pojem **EMR** (angl. *Electronic Medical Record*), ki se ga pogosto meša s pojmom EZZ (EHR). EMR se namreč osredotoča izključno na medicinske podatke, podatke namenjene za diagnosticiranje in zdravljenje, za razliko od EZZ, ki vsebuje tudi druge podatke, npr. podatke o pacientovem telesnem stanju (telesna višina). EMR predstavlja digitalno verzijo pacientovih medicinskih podatkov v ordinaciji zdravstvenega delavca [23]. Podatki iz EMR so dostopni le znotraj ene zdravstvene ordinacije in ni preprostega načina za njihov prenos k zdravniku z druge organizacije. Najpogosteje se, ko se pojavi potreba po prenosu podatkov iz EMR, le-te posreduje kar v tiskani obliki.

Pojem **PHR** (angl. *Personal Health Record*) je sistem, s katerim si posamezniki beležijo in med seboj delijo informacije o svojem zdravju ali zdravju oskrbovanca [24]. PHR sistem ni le statična podatkovna zbirka, ampak omogoča tudi združevanje in deljenje podatkov, ter uporabo skupnih programskih orodij [25]. Vsak posameznik je odgovoren za natančnost in verodostojnost svojih meritev. Možnost ima izbirati komu so njegove zdravstvene informacije dostopne. PHR se seveda ločuje od uradnih zdravstvenih kartonov, kreiranih s strani zdravstvenega osebja, je pa osebju v pomoč pri zdravstvenih preiskavah.

3.4.6 Podatkovni zapisi

Podatkovni zapisi (angl. *composition*) so neke vrste instance predlog. Vsak tak zapis je tudi validiran s pomočjo korespondenčne predloge. To pomeni, da se vsak podatek pred shranjevanjem v platformo običajno preko nekega API-ja (angl. *Application Programming Interface*) primerja, če je njegova vrednost ustrezna glede na omejitve nastavljene v referenčnem polju predloge in v primeru da ni, se vnos ne shrani. V tem primeru se na aplikacijo, ki je poslala zahtevo za shranitev zapisa, pošlje povratni HTTP zahtevek,

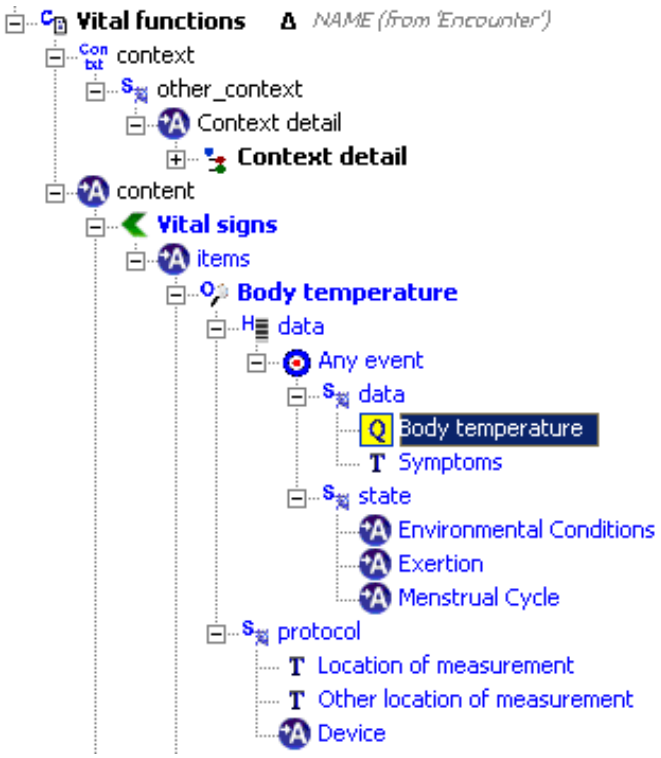
da je prišlo do napake pri shranjevanju in se uporabniku pokaže katere od podatkov je vnesel napačno. V nasprotnem primeru aplikacija od EHR strežnika prejme nov identifikator podatkovnega zapisa, ki ga lahko uporablja za nadaljnje delo v aplikaciji, oz. si ga shrani v podatkovno bazo. Slika 3.7 prikazuje zgradbo takega vnosa. Struktura vseh podatkovnih gradnikov na shemi je definirana na ravni referenčnega modela.



Slika 3.7 Struktura podatkovnega vnosa v EHR (angl. *composition*) [26].

Shemo s slike 3.7 je najlažje razložiti na podlagi primera (slika 3.8). Za primer smo vzeli predlogo vitalnih znakov. Prvi del predloge vsebuje obvezne kontekstne informacije. To so jezik vnosa, teritorij, časovni žig vnosa ter podatke o avtorju tega zapisa. Sledi vsebina vnosa. Znotraj te je sekcija (angl. *section*) vitalni znaki (angl. *vital signs*), ki vsebuje listo več meritev. Ena od njih je opazovanje (angl. *observation*) telesne temperature (angl. *body temperature*). Omenjeno opazovanje je zapisano v svojem arhetipu in predstavlja en zaključen primer uporabe. Znotraj enega opazovanja imamo lahko več izmerkov ali dogodkov (angl. *event*). Vsak od teh vsebuje izmerjene podatke

in stanje v katerem so bili ti podatki izmerjeni. Element telesna temperatura je tipa DV_QUANTITY (definiran v RM), ki vsebuje podatek o količini ter enoto merjenja. Opazovanje opiše tudi protokol na kakšen način smo opravili meritev, kje smo merili, s katero napravo itd.



Slika 3.8 Primer predloge (Vital Functions Encounter.oet) v urejevalniku Template designer podjetja Ocean Informatics.

3.4.7 AQL

Poizvedovalni jezik AQL (angl. *Archetype Query Language*) omogoča poizvedovanje po EZZ-jih. Temelji na poizvedovalnem jeziku SQL in pa na XPath-u (angl. *XML path language*). Njegova moč ne dosega moči jezika SQL, se mu pa vse bolj približuje. Omogoča izbiro rezultatov glede na OpenEHR strukturo, sortiranje, izbiranje glede na določen pogoj, paginacijo, omejevanje števila rezultatov in podobno. Omogoča tudi uporabo osnovnih agregacijskih funkcij za izračun minimalne, maksimalne in povprečne vrednosti ter štetje rezultatov (MIN,MAX,AVG,COUNT).

V pogojnem delu (WHERE) omogoča uporabo standardnih primerjalnih operatorjev (=, <, >, <=, >=, !=). Podpira tudi operatorja za obstoj določenega gradnika (EXISTS)

in za ujemanje vrednosti navedenega polja z vrednostjo iz seznama (MATCHES).

Za naslavljanje določenega elementa znotraj podatkovnega zapisa uporabljamo dolge XPath poti. Ker so te dokaj neberljive in jih je skoraj nemogoče sestaviti na pamet, si pri gradnji poizvedb pomagamo z orodji, s katerimi konstruiramo poizvedbe v uporabniku bolj razumljivi obliki, kasneje pa jih prevedemo v obliko z XPath-i, ki jo razumejo AQL procesorji.

Druga opazna razlika med SQL in AQL (vidna z zapisa 3.2) je uporaba izjave vsebovanja (CONTAINS). Izjava pomeni, da če katerikoli naslednik od trenutnega elementa vsebuje podatkovno strukturo, ki se ujema s podanim XPath-om, je zapis vrnjen in poslan v vnaprejšnjo obdelavo. S pomočjo te izjave lahko omejimo zapise na zelo splošen način, npr. glede na tip podatkovne strukture iz referenčnega modela ali pa zelo specifično npr. na konkretni arhetip. Tako v pogojnih stavkih, kot tudi v izjavi vsebovanja je možna uporaba logičnih operatorjev (AND, OR).

```
1  SELECT TOP 1 c
2  FROM EHR e[ehr_id/value='5beffd4a-287b-4922-88da-574bcf50318a']
3    CONTAINS COMPOSITION c[openEHR-EHR-COMPOSITION.encounter.v1]
4    CONTAINS SECTION sVitals[openEHR-EHR-SECTION.ispek_dialog.v1]
5  WHERE
6    c/name/value='Vital functions' AND (
7    EXISTS sVitals/items[openEHR-EHR-OBSERVATION.body_temperature-zn.v1] AND
8    sVitals/items[openEHR-EHR-OBSERVATION.body_temperature-zn.v1]/protocol[at0020]
9    /items[at0021.1]/value matches {'Ear canal', 'Skin'})
```

Zapis 3.2 Primer preproste AQL poizvedbe. TOP omeji število rezultatov na ena. V FROM delu povemo, da izbiramo podatke z navedenim EHR identifikatorjem. WHERE pogoj podatke omeji glede na tip zapisa na meritve vitalnih znakov. V tej poizvedbi torej izbiramo zapis meritve vitalnih znakov, ki vsebuje meritev telesne temerature (EXISTS del), katere lokacija meritve je bila opravljena na koži ali v ušesu (MATCHES del).

3.4.8 Primerjava z HL7

Za razliko od standarda HL7 je sistem postavljen po principih OpenEHR zelo šibko sklopljen, zmožnosti sistema pa so jasno znane [27]. OpenEHR specifikacija namreč določa kreiranje, shranjevanje in preiskovanje EHR podatkov in s tem omejuje razvijalce, kar je včasih ustrezno, včasih pa ne. Prav zaradi tega so mu nekateri naklonjeni, drugi pa ne. Podpira tako funkcionalno (dostopnost podatkov vsem), kot tudi semantično interoperabilnost (razumljivost podatkov za vse, tudi za računalnike), kar posledično pomeni, da je možno take podatke uporabiti za strojno učenje, odločitvene sisteme, planiranje, lokalizacijo itd. Standard HL7 se večinoma uporablja za sporočanje med kliničnim sistemom

in skupnimi centralnimi EHR sistemi. S prihodom verzije 3 tudi HL7 omogoča EHR in PHR (angl. *Personal Health Record*) interoperabilnost, saj tako kot OpenEHR uporablja referenčni model. Oba "standarda" potrebujeta podrobno definicijo kliničnih modelov. Ti vsebujejo kombinacijo terminologij, medicinskega znanja in strukture podatkov. V pripravi so že tudi orodja za pretvorbo OpenEHR arhetipov v predloge HL7 verzije tri.

4 Postopek razvoja vnosnih form

4.1 Cilji razvoja

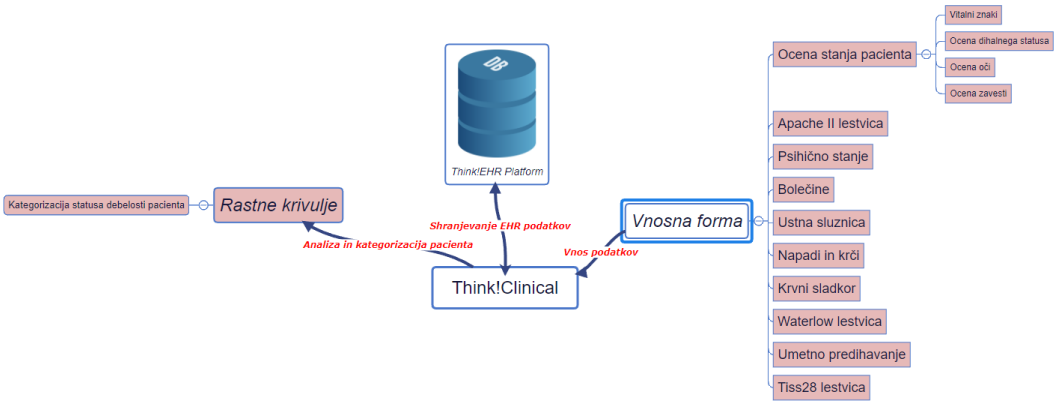
Kot že omenjeno je cilj pričujoče diplomske naloge izdelava vnosnih form s pomočjo temu namenjenih orodij, implementacija le-teh v klinični informacijski sistem *Think!Clinical* ter prikaz uporabe podatkov, vnesenih s temi formami, na praktičnem problemu. Posredni cilj je prikazati kako hiter je lahko razvoj preprostih polno-funkcionalnih vnosnih form z ustreznimi orodji ter platformo *Think!Clinical*, ter kakšne napredne opcije ponujajo ta orodja in platforma. V današnjem svetu je namreč agilni razvoj ter hitra odzivnost podjetij na zahteve uporabnikov ključnega pomena.

Jim Highsmith je v svoji knjigi leta 2000 zapisal, da biti agilen pomeni, da lahko hitro dostavljaš produkte ter jih hitro in pogosto spreminjaš [28]. Agilne metode razvoja se običajno precej razlikujejo v načinih izvedbe agilnih metod, a si delijo skupne lastnosti, kot na primer iterativni razvoj, interakcijo ter komunikacijo med naročnikom in razvojno ekipo ter minimizacijo stroškov. Iterativni razvoj z majhnimi iteracijami omogoča razvojni ekipi hitro prilagajanje in spreminjanje zahtevanih funkcionalnosti. Delo v manjših ekipah z rednimi dnevnimi sestanki omogoča dobro komunikacijo znotraj ekipe in s tem

pospeši delo ter zmanjša možnost nesporazumov. Poznamo veliko agilnih metod razvoja, kot na primer "scrum", ekstremno programiranje, programiranje v parih, a na tem mestu se ne bi spuščali v podrobnosti le-teh metod.

Vnosne forme izdelane v sklopu pričujočega diplomskega dela obsegajo področje opazovanj stanja pacienta. Gre za vnosne forme vitalnih znakov, oceno dihalnega statusa, oči, zavesti, psihičnega stanja, bolečine, oceno stanja ustne sluznice, napadov in krčev, krvnega sladkorja, oceno ogroženosti pacienta (angl. *waterlow*), oceno stanja pacienta ob umetnem predihavanju ter lestvico nujnosti (angl. *TISS28*). Ker je teh vnosnih form kar precej, se v pričujočem diplomskem delu osredotočamo na tiste zahtevnejše za implementacijo. To so forme vitalnih znakov, ocene dihanja ter ocena stanja ustne sluznice.

Drugi praktični del pričujočega diplomskega dela je nadgradnja funkcionalnosti kategorizacije debelosti mladostnikov. To je funkcionalnost, ki zdravnikom omogoča spremljanje razvoja otrok mlajših od dvajset let, ter jih opozarja na možne nepravilnosti. V sklopu diplomskega dela je bila razvita tudi funkcionalnost izvoza podatkov vnesenih z vnosnimi formami vitalnih znakov ter ocene zavesti v PDF dokumente za poročanje. Prikaz celotnega prispevka diplomskega dela je viden na sliki 4.1, na kateri je prispevek diplomskega dela označen z rdečo barvo.



Slika 4.1 Prikaz prispevka (označeno z rdečo barvo) v klinični informacijski sistem *Think!Clinical* opravljenega v okviru pričujočega diplomskega dela.

4.2 Uporabljene tehnologije

Med izdelavo praktičnega dela te diplome smo se spoznavali z različnimi tehnologijami, programskimi jeziki in orodji za razvoj. Nekatera od teh so že bila deležna opisa v prejšnjih razdelkih. Prav je, da se na tem mestu spoznamo še z ostalimi.

4.2.1 AngularJS

AngularJS je ogrodje za razvoj spletnih aplikacij, zasnovan po principu MVC (angl. *Model View Controller*). Funkcionalnosti AngularJS aplikacije so razbite na več različnih komponent. Aplikacija se deli na tri osnovne dele:

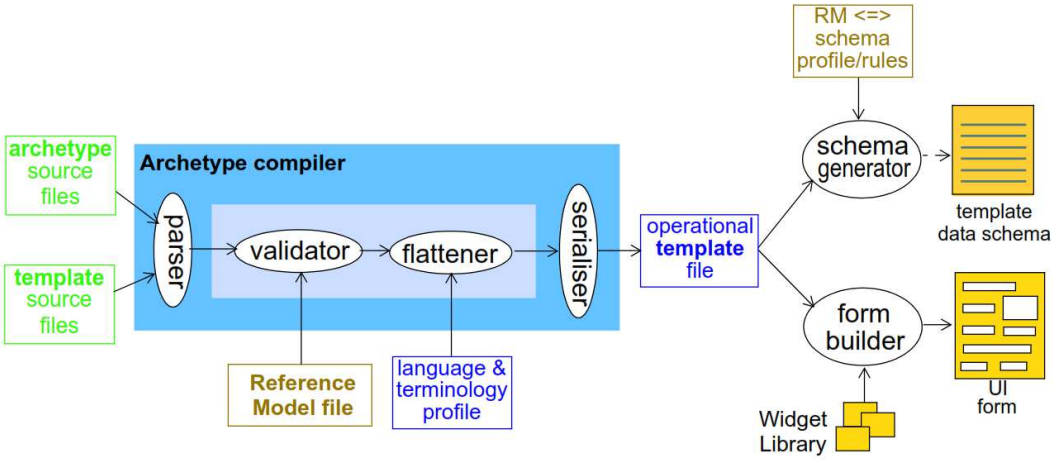
- model (angl. *model*), ki nosi podatke in logiko;
- pogled (angl. *view*), ki nosi vizualno predstavitev podatkov;
- "controller", ki povezuje oba zgornja dela.

Model aplikacije skrbi za upravljanje s podatki posamezne komponente. Vrednosti z modela so direktno vezane na HTML DOM elemente v pogledu. Ob spremembi modela se tako spremeni tudi vrednost na pogledu. Temu strokovno rečemo "data binding". Ob spremembi modela običajno želimo izvesti določeno akcijo. Za spremljanje sprememb določene spremenljivke uporabljamo poslušalce (angl. *watcher*). Ko le-ta zazna spremembo na modelu pokliče funkcijo, ki smo jo določili za poslušalca in izvede željeno akcijo (preračun podatkov, validacija itd.).

4.2.2 Načrtovalec vnosnih form - "form builder"

Kot je vidno iz slike 4.2, je načrtovalec vnosnih form (angl. *form builder*) namenjen izdelavi vnosnih form. Eden on njih je tudi "EHR Explorer", produkt podjetja Marand, tako kot tudi "form renderer", ki skrbi za izris forme, narejene s tem orodjem. Nabor podatkov na formi je omejen na podatke, ki so prisotni na operativni predlogi, ki jo načrtovalec izbere, ko začne z izdelavo. Postopek razvoja vnosne forme je opisan v razdelku 2.5.1.

Ko vnosno formo v urejevalniku shranimo, se ustvarijo štiri JSON datoteke. Prvi in najpomembnejši je opis forme (angl. *form description*). Ker ima vnosna forma način za prikaz ter način za urejanje podatkov, sta opisa forme dva, za vsakega od tipov eden. Poleg tega ena JSON datoteka vsebuje tudi opis postavitve polj na formi, ki pa se v



Slika 4.2 Razvojni krog razvoja vnosne forme od arhetipov do uporabniškega vmesnika [29].

trenutni verziji generatorja še ne upoštevajo. V zadnji datoteki so "form dependancies", ki omogočajo implementacijo preproste logike kar z grafičnim urejevalnikom. Omogočajo izvedbo več akcij na določen pogoj. Pogoji so prazna vrednost, izbrana vrednost, enakost ter neenakost vrednosti v polju. Na izpolnjen pogoj lahko prožimo akcije prikaži/skrij, omogoči/onemogoči ter nastavi/izbriši oz. ponastavi vrednost na privzeto.

Anotacija

Anotacije so pari ključa in vrednosti, ki služijo, da lahko na ravni predloge dodajamo dodatne metapodatke na posamezna polja. Dodamo jih lahko na poljuben element [30]. S pomočjo anotacij lahko npr. dodajamo dodatne prevode, ki jih nato uporabimo za prikaz na vnosni formi.

Anotacije so ključne za grajenje naprednih vnosnih form. Ideja je, da bi s pomočjo anotacij nadomestili določena označena polja s poljubno obogateno vsebino, oz. jih razširili z dodatno vsebino. Na polje, ki ga želimo obdelati, dodamo anotacijo, ki nam pove, katera funkcija naj se pokliče ob generiranju vnosne forme in zamenja oz. razširi vsebino izbrane komponente na formi. Glede na to, kakšne vrste je ta funkcija, jih delimo na razširitve (angl. *extension*) in posebne komponente (angl. *custom component*). Tak način dela bi moral omogočati, da bi lahko razvijalci večkrat uporabljali iste skupne funkcije zbrane v nekem skupnem repozitoriju ter s tem pospešilo razvoj tovrstnih form.

Ni pa nujno, da te funkcije spreminjajo le izgled vnosnih form, ampak lahko le njihovo

funkcionalnost. Primer take uporabe funkcij bi bil, če bi razvijalec na vsebnik, ki vsebuje polja teža, višina in ITM (indeks telesne mase), dodal anotacijo "funkcija" z vrednostjo "izracunajBMI" in ta funkcija bi poskrbela za poslušanje sprememb vrednosti v poljih višine in mase, ter vrednost zapisala v polje ITM.

Oznaka

Oznake (angl. *tags*) imajo podobno vlogo kot anotacije, a ne vsebujejo dvojic ključ vrednost, temveč le vrednosti. V "form builderju" služijo označevanju polj, da jih lahko potem na generirani formi preprosteje najdemo s pomočjo te oznake. Poleg tega služijo tudi označevanju polja z različnimi stanji in posebnostmi. Primeri oznak so sledeči:

- Oznaka "multi": polje bo na generirani formi imelo omogočeno dodajanje več instanc;
- Oznaka "ignoreForContainsValue": polje se ne upošteva pri preverjanju, če ima forma vnesene vrednosti;
- Oznaka "noRender": za polje se ne ustvari DOM element, ampak se ustvari le njegov podatkovni model;
- Oznaka "viewMode": polje se prikaže v prikazovalnem načinu, čeprav je forma v načinu urejanja.

4.2.3 Generator vnosnih form - "form renderer"

Generator vnosnih form skrbi za dinamično grajenje vnosnih form glede na vhodne podatke v JSON obliki. Kot vhodne podatke sprejema opis forme (angl. *form description*), ki je zgrajen z "form builder-jem". Omenjeno datoteko prebere in se iterativno premika skozi seznam elementov v njej. Za vsakega prebere njegov tip in ga prikaže z ustreznim grafičnim elementom. Za prikaz uporablja zunanjo knjižnico grafičnih gradnikov.

Zapis 4.1 predstavlja primer predstavitve polja na formi (del opisa forme (angl. *form description*)). Poleg tipa elementa ta opis vsebuje tudi oznako, kaj element predstavlja, ki jo običajno uporabimo kot "labelo" pred poljem. Definirana je številnost polja, ki nam pove minimalno in maksimalno koliko vnosov enega podatka imamo lahko na isti formi. Sledijo anotacije ter oznake. Prisotna sta tudi dva identifikatorja, da lahko polje lažje najdemo. V polju "inputs" so podatki, ki opisujejo delovanje posameznega vnosnega

polja (za vrednost ali enoto). Tu so prisotni tip polja, podatki za validacijo posameznega polja (minimum, maksimum), natančnost podatka (za številčne), nabor vseh možnih vrednosti itd. V opisu polja je specificirana tudi njegova privzeta vrednost (če je ta določena). Opis vsebuje konfiguracijo pogleda (angl. *view config*), ki nosi podatke o velikosti, umestitvi polja, posebnih oznakah, zunanjih šifrantih itd.

Vse te podatke "form renderer"upošteva in jih uporabi pri grafični predstavitvi. Za vsak element v opisu forme kreira njegovo HTML predstavitev in "angular" model, mu določi validacijo, privzete vrednosti in enote, velikost ter vse druge potrebne parametre. Kot že omenjeno v predhodnem razdelku 4.2.2, "form renderer"omogoča tudi uporabo posebnih in razširjenih komponent. Če je na elementu z anotacijo navedena funkcija, jo "form renderer"pokliče in nato funkcija poskrbi, da se pogled razširi oz. zamenja s posebno komponento. "form renderer" funkciji posreduje "formApi", ki ponuja nabor osnovnih funkcij s katerimi manipuliramo s formo in jo spreminjamo. Te funkcije omogočajo spreminjanje AngularJS modela posamezne komponente, ki se nato odraža na spremembi prikaza vnosne forme.

```
{
  "name": "Heart rate",
  "localizedName": "Heart rate",
  "rmType": "DV_QUANTITY",
  "nodeId": "at0004",
  "min": 0,
  "max": 1,
  "localizedNames": {
    "en": "Heart rate",
    "sl": "Pulz"
  },
  "localizedDescriptions": {
    "en": "The rate of the heart in beats per minute.",
    "sl": "Stevilo sr nih utripov na minuto"
  },
  "annotations": {
    "default": "L10n={sl=Pulz}"
  },
  "aqlPath": "/content[openEHR-EHR-SECTION.ispek_dialog.v1,'Vital signs']/items[
      openEHR-EHR-OBSERVATION.heart_rate-pulse-zn.v1]/data[at0002]/events[at0003]/
      data[at0001]/items[at0004,'Heart rate']/value",
  "inputs": [2],
  "formId": "vital_functions/vital_signs/pulse/any_event/heart_rate",
  "viewConfig": {
    object
  }
```

```
25 }
```

Zapis 4.1 Primer JSON opisa za polje na formi. Zaradi dolžine zapisa sta polji "viewConfig" in "inputs" skrčeni.

4.3 Razvoj vnosne forme vitalnih znakov

Razvoj se je začel z nalaganjem operativne predloge na ustrezni EHR strežnik. Ta korak je obvezen, kajti celotna logika forme, od shranjevanja, validacije in prikaza, temelji na omejitvah navedenih v operativni predlogi. Sledilo je dodajanje ustreznih polj na formo. Naša naloga je bila prenoviti stare forme narejene v "javanski" knjižnici "Swing". Tako smo lahko ustrezni nabor polj izbrali kar iz stare forme oz. iz kode v Javi. Ker se za prikaz na različnih produkcijskih okoljih uporablja različen nabor polj in nekoliko drugačna logika, smo izdelalo dve različni osnovni formi in sicer eno bolj podrobno za *klinični oddelek za otroško kirurgijo in intenzivno terapijo (KOOKIT)* ter eno za preostala okolja.

Ker se iz osnovne vnosne forme preko gumbov odpirajo druge forme, ki omogočajo izbiro in vnos dodatne razširjene vsebine osnovne forme, smo se v drugem koraku lotili izdelave le-teh. Gre za forme vitalnih znakov (razširjena forma z zavihki), formo za vnos dihalnega statusa, formo za vnos stanja zavesti, ter formo za oceno stanja oči. Za vsakega od polj na vnosni formi je bilo potrebno nastaviti privzeto velikost, urediti prevode, po potrebi urediti omejitve nabora vrednosti, nekatera izračunljiva polja (npr. indeks telesne mase ali površino kože) označiti kot le za branje ("readonly" nastavitev) itd.

Naslednji korak je bil na forme dodati pravila - "form dependancies". To so preprosta navodila za generator, ki omogočajo izvedbo enostavne logike na vnosni formi (več v razdelku 4.2.2) in s tem programer prihrani pri številu vrstic kode. Primer uporabe teh pravil bi bil sledeči scenarij: Vnos telesne temperature poleg vnosa izmerjene višine omogoča tudi vnos lokacije meritve. Validacija vnosa telesne temperature je taka, da je v primeru vnesene lokacije le-ta obvezen podatek. Lokacija meritve namreč spada pod tip vnosa "protocol", zato zahteva prisoten vsaj en podatek iz podatkovne sekcije. Omenjeni scenarij rešimo precej preprosto. Ko je vrednost v polju temperature prazna, je polje lokacije onemogočeno. V primeru vnosa se omogoči polje lokacije. Če se vrednost temperature ponovno briše, je potrebno vrednost v polju lokacije pobrisati in polje ponovno onemogočiti. Podobno velja tudi za polje "dodaten opis", ki tudi spada k meritvi telesne temperature. Vse to smo rešili z uporabo dveh pogojnih stavkov v pravilih na formi.

Prikaz, kako se nastavlja tovrstna pravila, je viden na sliki 4.3.

Slika 4.3 Prikaz urejanja pravil v orodju "form builder".

4.3.1 Specialne funkcije

Sledila je implementacija glavne logike. Na krovni element forme smo dodali anotacijo "function" in ji kot vrednost dodali ustrezno ime. Vsaka kompleksna vnosna forma je v aplikaciji *Think!Clinical* predstavljena kot samostojni Javascript razred - "view". To je Javascript objekt s spremenljivkami in funkcijami, ki podpira dedovanje ter abstrakcijo, definiran s pomočjo ogrodja, ki je last podjetja. Primer takega razreda je prikazan v zapisu 4.2.

```
1  Class.define('app.views.ehrform.plugins.vitals.VitalsEhrFormView',
2          'app.views.ehrform.plugins.common.AbstractEhrFormView', {
3    Constructor: function()
4    {
5      this.callSuper();
6    },
7    /**
8     * @Override
9     * @returns {[]}
10    */
11   getDefaultCustomFunctionConfigs: function()
12   {
13     var namespace = this.getBindingFunctionNamespace();
14     return [
15       {
16         bindingFunctionNamespace: namespace,
17         bindingFunctionName: "VitalsRoot",
18         bindingFunctionClass: app.views.ehrform.plugins.vitals.VitalsCustomFunction,
19         bindingFunctionClassConfig: null
```

```
20        },
21        {
22          bindingFunctionNamespace: namespace,
23          bindingFunctionName: "VitalsWeightIcuCustomFunction",
24          bindingFunctionClass: app.views.ehrform.plugins.vitals.
                VitalsWeightCustomFunction,
25          bindingFunctionClassConfig: {view: this, data: {showPercentile: true,
                showOpenDetailsButton: false}}
26        }
27      ];
28    }
29  });
```

Zapis 4.2 Primer Javascript razreda, ki predstavlja samostojno zaslonsko masko. Kot tipični razredi v kateremkoli objektnem jeziku ima ta razred tudi konstruktor. V primeru ta konstruktor kliče konstruktor razreda, ki ga razširja (angl. *extends*) (v Javi this.super()). Funkcija getDefaultCustomFunctionConfigs() vsebuje konfiguracijo funkcij za posamezno formo. Ta konfiguracija povezuje anotacijo na formi z ustrezno "javascript" funkcijo, ki se izvrši ob inicializaciji forme. Preko konfiguracije lahko v funkcijo nesemo dodatne podatke (npr. trenutni "view" ter ostale kontrolne podatke).

Vsak tak razred poskrbi, da se ob inicializaciji registrirajo vse funkcije, ki jih forme potrebujejo. Te funkcije lahko razširjajo logiko delovanja vnosnih form (izračuni raznih ocen stanja), ali pa spreminjajo privzeti izgled vnosne forme (v tem primeru jim rečemo posebne komponente). Funkcije se registrirajo na globalni "window" objekt v spletnem brskalniku in sicer z ustrezno imensko domeno (angl. *namespace*) kateri sledi ime funkcije. Ogrodje nato ob inicializaciji forme glede na vrednost anotacije "function" na formi izvede ustrezno funkcijo, ki dobi kot vhodni podatek "form API" ter "form model", ki omogočata popolno manipulacijo nad objekti na vnosni formi. Običajno je, da v podjetju uporabljamo eno glavno funkcijo, ki skrbi za logiko na celotni formi in več posebnih komponent. Glavni del pričujoče diplomske naloge je implementacija teh funkcij, ki so "možgani" vnosne forme.

Za krajši prikaz, kako take funkcije delujejo, smo si izbrali del krovne funkcije na formi vitalnih znakov, ki preračunava površino kože. Primer je prikazan v zapisu 4.3. Kot je običajno pri razredih, ki so razširjeni z drugim razredom, se najprej pokliče konstruktor starševskega razreda. Form API se pridobi s pomočjo "getter" funkcije, ki je definirana v starševskem razredu. Ker so poti do polj (oz. drugače rečeno njihovi identifikatorji) shranjene v drugem razredu, si jih shranimo v spremenljivko. EhrFormHelpers (9. vrstica zapisa 4.3) je razred splošnih funkcij za delo s polji na formah in je nastajal tekom razvoja HTML vnosnih form. Omogoča dodajanje poslušalcev sprememb na polja, iskanje elementa po identifikatorju, pridobivanje vrednosti glede na tip polja, skrivanje,

onemogočanje in podobno.

Funkcija "watchMagnitudePoint" (11. vrstica zapisa 4.3) nastavi na polje telesne višine poslušalec (angl. *watcher*) v AngularJS okolju in ob vsaki spremembi izvede navedeno funkcijo. Ker je potrebno vrednost izračunati v primeru, da se spremeni višina, teža ali formula za izračun, se poslušalec nastavi na vsa tri polja. Nato preko "helpers" funkcije pridobimo vrednosti iz polj, ter preverimo, če so vse tri definirane in če so, izračunamo novo vrednost, v nasprotnem primeru pa vrednost pobrišemo. Funkcija za izračun površine kože uporablja skupni razred (MedicalKnowledgeUtils.js - 19. vrstica zapisa 4.3), ki vsebuje različne formule ter splošne metode za različne izračune, kot so npr. indeks telesne mase, idealno telesno težo, površino kože, količino izdihanega zraka in podobne.

```
1  Class.define('app.views.ehrform.plugins.vitals.VitalsCustomFunction',
2          'app.views.common.ehrform.AppEhrFormCustomFunction', {
3    Constructor: function()
4    {
5      this.callSuper();
6      var formApi = this.getFormApi();
7
8      var PATHS = app.views.ehrform.plugins.vitals.Paths.getPaths();
9      var EhrFormHelpers = app.views.ehrform.Helpers;
10
11     EhrFormHelpers.watchMagnitudePoint(formApi, PATHS.HEIGHT, this.
           recalculateSkinSurface);
12     EhrFormHelpers.watchMagnitudePoint(formApi, PATHS.WEIGHT, this.
           recalculateSkinSurface);
13     EhrFormHelpers.watchCodePoint(formApi, PATHS.FORMULA_NAME, this.
           recalculateSkinSurface);
14     this.recalculateSkinSurface();
15   },
16   recalculateSkinSurface: function()
17   {
18     var EhrFormHelpers = app.views.ehrform.Helpers;
19     var MedicalKnowledgeUtils = app.views.ehrform.plugins.common.MedicalKnowledgeUtils
           ;
20     var PATHS = app.views.ehrform.plugins.vitals.Paths.getPaths();
21
22     var weightKg = EhrFormHelpers.getMagnitudeValueOrZero(formApi, PATHS.WEIGHT);
23     var heightCm = EhrFormHelpers.getMagnitudeValueOrZero(formApi, PATHS.HEIGHT);
24     var formulaVal = EhrFormHelpers.getCodeValueOrNull(formApi, PATHS.FORMULA_NAME);
25
26     if (weightKg > 0 && heightCm > 0 && formulaVal !== null)
27     {
28       var formula = PATHS.FORMULA_MAP[formulaVal];
29       var surface = MedicalKnowledgeUtils.calcBSA(formula, heightCm, weightKg);
```

```
30        EhrFormHelpers.setMagnitudeValue(formApi, PATHS.BODY_SURFACE, surface);
31      }
32      else
33      {
34        EhrFormHelpers.setMagnitudeValue(formApi, PATHS.BODY_SURFACE, null);
35      }
36    }
37  });
```

Zapis 4.3 Okrnjena funkcija z logiko za formo vitalnih znakov.

Na podoben način je potrebno preračunati tudi indeks telesne mase, idealno telesno težo, pusto telesno maso, pulzni in srednji krvni pritisk ter izvajati dodatno logiko skrivanja in onemogočanja na nekaterih poljih, kjer tega ni mogoče izvesti s pomočjo "form dependency"-jev. Primer so polja z več možnimi vrednostmi (oznaka "multi"), ki glede na izbrano opcijo prikazujejo, ali skrivajo druga polja.

Podobni preračuni so potrebni tudi na vnosni formi za oceno dihalnega statusa. Tu je potreben izračun predvidene količine izdihanega zraka (angl. *Peak Expiratory Flow* - PEF). Ko se za pacienta vnese količino izdihanega zraka, se mu vrednost izračuna na podlagi te meritve in njegove telesne višine. Le-te pacientu ni potrebno vnesti v sklopu istega vnosa, zato se iz aplikacijskega strežnika pridobi zadnja meritev (več v zapisu 4.4).

```
1  Class.define('app.views.ehrform.plugins.respiration.RespirationHelpers', 'app.views.
       ehrform.Helpers', {
2    statics: {
3      receivePatientHeightRespirationData: function(view, callback)
4      {
5        var observationsParams = {
6          patientid: view.getActivePatientId()
7          };
8        var observationsUrl = view.getViewContextUrl() + "/patient/state";
9        view.sendGetRequest(observationsUrl, observationsParams,
10         function(data)
11         {
12           callback(data);
13         });
14     }
15   }
16 });
```

Zapis 4.4 Statična funkcija se pokliče ob inicializaciji pogleda za oceno dihalnega statusa in s pomočjo pacientovega identifikatorja vrne zadnje podatke o njegovem stanju (teža, višina), ki nato v podani "callback" funkciji nastavi vrednost višine, ki se nato uporablja pri izračunih.

4.3.2 Posebne komponente

Ker se iz osnovne forme vitalnih znakov odpirajo druge forme ob kliku na gumb ob strani (slika 4.4), je bilo potrebno implementirati "custom extended component"-o, ki to funkcionalnost omogoča. Funkcija, ki je z anotacijo vezana na element forme, dobi referenco na ta element, in mu doda vsebino na desni (v tem primeru gumb). Element ima tudi anotacijo, katere vrednost pove, katero podformo je potrebno odpreti, ter kateri zavihek na formi naj bo izbran, ko se forma odpre. Na krovnem pogledu vitalnih znakov je funkcija, ki mapira anotacijo v podatke potrebne za odpiranje razširjene forme. Ta funkcija je podana kot vhodni parameter te posebne komponente, zato jo lahko ob kliku na gumb uporabi ter odpre ustrezno podformo.

Slika 4.4 Gumbi za odpiranje razširjenih vnosnih form predstavljajo primer razširjene komponente na formi. Na sliki sta vidni dve posebni komponenti. Prva preprostejša je na polju telesna temperatura in polje razširja z gumbom za odpiranje podrobnejše forme. Druga posebna komponenta na polju telesna teža vsebuje razširjeno polje teža v gramih, ki omogoča natančnejši vnos ter hkrati prikazuje percentil teže (100P na desni). Percentil teže za posameznika pove, kakšna je teža otroka (starost do 20 let) glede na ostalo populacijo. Petdeset procentov tako predstavlja povprečje glede na njegovo starostno skupino. Tudi izračun izdelava izračuna percentilov na strani odjemalca je bil del naše naloge. Percentile se računa za telesno težo, višino, indeks telesne mase ter obseg glave. Za vsako od teh se iz množice zbranih podatkov s pomočjo linearne interpolacije ter podatka o meritvi ter starosti pacienta (na dan meritve) izračuna, če je pod ali nadpovprečen glede na starost.

Vsaka forma nosi podatke, ki so trenutno zapisani v modelu forme. Ti podatki so strukturirani v obliki JSON. Osnovna forma seveda ne vsebuje vseh podatkov, ki jih vsebujejo razširjene. Zato je potrebno na nivoju osnovne forme hraniti združene vrednosti vseh polj. Ob vsakem prehodu med razširjeno in osnovno formo je potrebno prenesti spremenjene vrednosti oz. dodati nove v skupni objekt vrednosti ter jih nastaviti na polja vnosne forme. Prav tako se ob vsakem prehodu izvede validacija na formi, ki jo izvede funkcija na generatorju form (ki jo pokličemo) in vrača dve vrednosti: veljavna oz. neveljavna forma. Glede na to uporabnika ustrezno opozorimo, da so vrednosti napačno vnesene, polja na katerih pade validacija pa generator avtomatsko obarva rdeče.

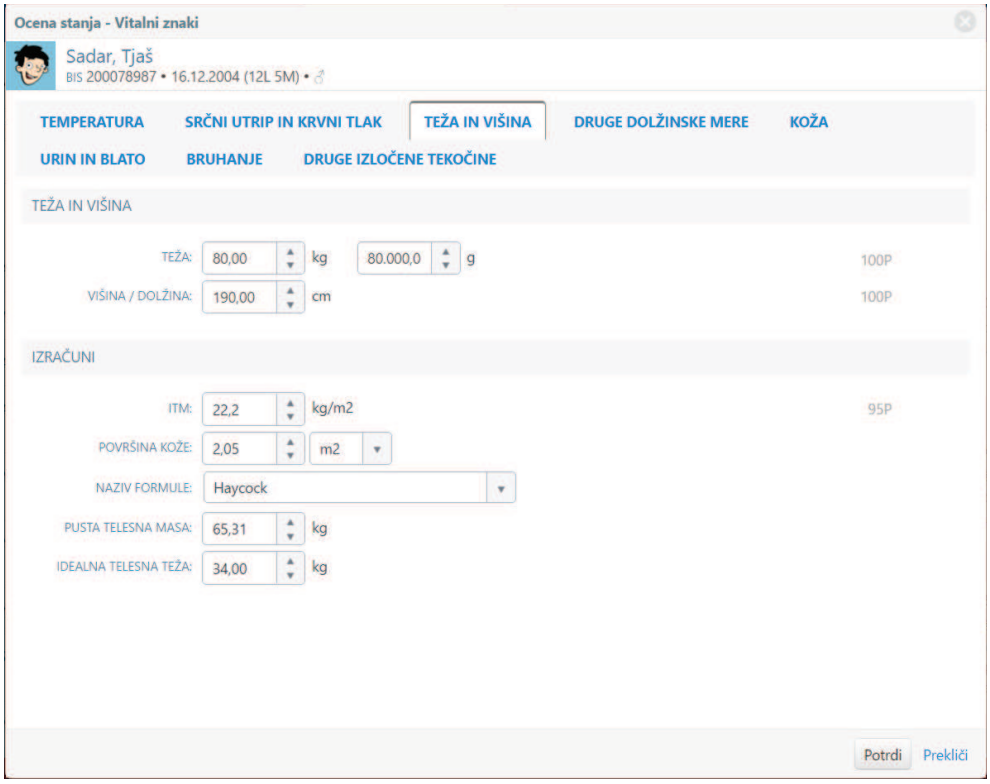
Validacijo je potrebno obvezno izvajati na vsakem koraku, da ne bi uporabnik na koncu ob shranjevanju na glavni vnosni formi naletel na pokvarjene podatke, ki jih niti ne bi mogel več popraviti.

4.4 Shranjevanje form

Običajna vnosna forma se shrani na tak način, da se s funkcijo, ki jo ponuja generator form, pridobi JSON objekt vrednosti, se jih validira, ter po potrebi še preveri, če je sploh kakšna vrednost vnesena, saj nočemo podpirati shranjevanja praznih podatkovnih zapisov. Forma ocene stanja vitalnih znakov zahteva nekoliko več logike ob shranjevanju, od preprostih eno-nivojskih form.

Ob shranjevanju najprej izvedemo dodatno logiko, ki poskrbi, da se bodo vrednosti pravilno shranile. Spet so težave zaradi različnih tipov podatkovnih vnosov (state/data tipa). Validacija ob shranjevanju namreč pada, če shranimo neko vrednost sekcije v "state", ne shranimo pa nobenega podatkovnega vnosa tipa "data", saj OpenEHR specifikacija ne dopušča shranjevanja stanja brez podatkovnih vnosov (tak primer je opisan v opisu sliki 3.5). Že v podpoglavju 4.3.1 smo opisali, kako se računa površina kože (forma na sliki 4.5). Problem nastane, če shranjujemo le formulo za izračun površine kože, ne pa izračunane vrednosti površine. Težava se pojavlja v primerih, če uporabnik vnese svojo težo, višino ter izbere formulo, nato pa pobriše težo ali višino oz. oboje. V tem primeru uporabniku formule nočemo takoj izbrisati, saj se je mogoče le zmotil pri vpisovanju in vseeno hoče izvesti izračun. Izbris formule bi bil v tem primeru zanj zelo moteč. Ta korak zato raje storimo ob shranjevanju, torej preverimo če obstaja vrednost površine kože in v primeru da ne, vrednost formule izbrišemo.

Kot smo že omenili v sklopu 4.3.2, se ob vsakem prehodu med osnovno in razširjeno formo izvede validacija, ob tem pa tudi nastavimo vrednost v poseben objekt, ki za vsako razširjeno formo nosi podatek, če forma vsebuje vrednosti. Ob tem se hkrati prenesejo vrednosti aktivne forme v skupni objekt vrednosti na krovni formi. Pred shranjevanjem se ponovno prenesejo vrednosti z osnovne forme v združen objekt vrednosti. Nato sledi validacija. Ker je forma zasnovana kot "plugin view", kar omogoča, da jo vključimo znotraj poljubnega pogleda v aplikaciji, lahko s podanim argumentom povemo, če hočemo omogočati shranjevanje praznih vrednosti. Če je ta vrednost omogočena, formo shranimo v vsakem primeru, drugače pa le v primeru, če katera od razširjenih form vsebuje vredno-



Slika 4.5 Zaslonski posnetek vnosne forme vitalnih znakov (razširjena forma z zavihki) odprte na zavihku teža in višina.

sti, ali če so vrednosti prisotne na osnovni formi. Sledi še zaključna validacija osnovne forme (razširjene smo validirali sproti) ter pošiljanje POST zahtevka na strežnik, ki vsebuje celotni objekt združenih vrednosti. Podatki se nato na API-ju platforme validirajo in uporabniku je ob uspešnem vnosu poslan identifikator vnosa "compositionUid", ki ga nato uporablja za spreminjanje oz. pregledovanje vnosa, v primeru napake pri validaciji pa dobi ustrezno napako.

4.5 Ocena stanja ustne sluznice - izdelava vnosne forme

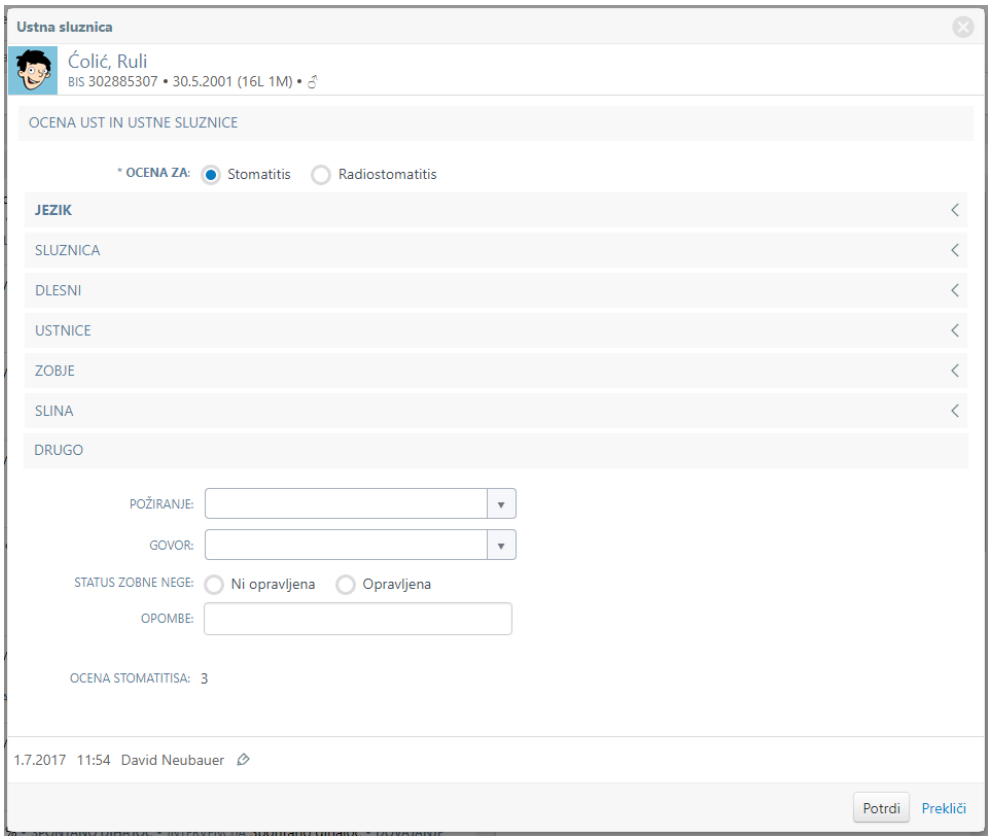
Podobno kot pri formi za oceno stanja vitalnih znakov pacienta, se je tudi razvoj forme za oceno stanja ustne sluznice začel z razvojem forme v "form builder"-ju. Na vnosno formo smo dodali polja, ki so bila že prisotna na stari "swing" formi. Na krovni element forme smo ponovno dodali anotacijo, ki je ob inicializaciji forme nastavila funkcijo za izračun ocene.

4.5.1 Implementacija logike vnosne forme

Prvo od polj na formi je polje, s katerim si uporabnik izbere, katero oceno želi izračunati. Tako oceno "stomatitisa", kot tudi oceno "radiostomatitisa" računamo po isti formuli. Kot je vidno iz slike 4.6, je forma razdeljena v več skupin podatkov (jezik, sluznica, dlesni, ustnice, zobje, slina, drugo). Vsaka od skupin jezik, sluznica, dlesni in ustnice vključuje polja ocene rdečine, razjed in občutka pekočega oz. zbadanja. Vseh teh dvanajst polj se uporablja za izračun ocene stomatitisa ali radiostomatitisa. Na vsakega od teh polj se ob inicializaciji nastavi poslušalca, ki ob spremembi preračuna vrednost ocene, le-ta pa se nato shrani v EZZ pacienta.

4.5.2 Design forme

Kot je vidno iz slike 4.6, je forma ustne sluznice razdeljena na več skupin podatkov, ki jih prikazujemo z "accordion"-om. Ker "form renderer" ne podpira prikaza z "accordionom", je bilo potrebno ta problem rešiti s posebnimi komponentami. Vsaka od skupin podatkov je predstavljena s svojim vsebnikom, kot običajno. Ta kontejner vsebuje oznako skupine, katere podatke nosi. Za vsako od teh skupin je na formo dodano posebno polje, ki na vnosni formi predstavlja zgornjo pasico vsake od skupin. Skrbi za zapiranje in odpiranje vsake od skupin podatkov ter prikazovanje stanja, če je kateri od podatkov v skupini že



Slika 4.6 Zaslonski posnetek vnosne forme za vnos stanja ustne sluznice. Iz forme je vidno, da je že vnesen eden od podatkov iz sklopa ocene jezika in ustrezno izračunana ocena stomatitisa.

vnesen. Vsako od posebnih polj ima na formi dodano anotacijo, ki mu pove, kateri od skupin pripada.

Posebno polje je privzeto predstavljeno kot vsako izmed vnosnih polj. To polje je nato nadomeščeno s posebno komponento, ki je definirana preko anotacije na formi. Ta posebna komponenta spremeni prikaz posebnega polja in ga prikaže kot zgornjo pasico skupine.

V "javascript" kodi posebne komponente najprej preberemo anotacijo, ki pove kateri skupini pripada posebna komponenta. Iz modela skupine preberemo njeno ime in ga izpišemo na levi strani, na desno stran pa dodamo indikator, ki prikazuje ali je skupina odprta ali zaprta. Postavitev rešimo z dodajanjem novih DOM elementov, ter kaskadnimi slogi (CSS). Na "form model" skupine dodamo poslušalca sprememb, ki nam ob vsaki spremembi vrednosti v izbrani skupini kliče funkcijo, ki preveri, če skupina vsebuje vrednosti in glede na to nastavi slog pisave besedila skupine na krepko (če je katera od vrednosti vnesena), ali na običajno debelino. S to ne preveč kompleksno rešitvijo smo razširili funkcionalnost "form renderer"-ja in za ta specifični primer formo zelo polepšali.

4.6 Uporaba EHR podatkov - rastne krivulje

Drugi praktični del pričujočega diplomskega dela je implementacija logike za kategorizacijo "posameznikove debelosti". Dana funkcionalnost je bila predhodno že implementirana v aplikaciji *Think!Clinical*, a ni vračala zadovoljivih rezultatov. Kategorizacijo je izvajala na podlagi percentila teže posameznika.

Percentil je mera, ki prikazuje kakšen je razvoj posameznika, glede na ostale vrstnike. Računa se ga za osebe mlajše od dvajset let, za telesne karakteristike teže, višine, indeksa telesne mase ter obsega glave. Če je posameznikov percentil enak 50% to pomeni, da je povprečen gleda na svoje vrstnike. Bolj kot odstopa od svojih vrstnikov, bolj se vrednost percentila oddaljuje od petdeset.

4.6.1 Problem obstoječe kategorizacije

Težava obstoječe kategorizacije je bila, da je v primeru, če je bil pacient višje rasti od povprečja in imel posledično tudi več kilogramov od povprečja, ga je sistem opredelil kot predebelega. Enako velja tudi za osebe nižje rasti, katerim je sistem napovedal podhranjenost. Nov način kategorizacije na podlagi percentila ITM, ki upošteva razmerje med

težo in višino, je to težavo v večini primerov rešil, pojavili pa so se novi implementacijski izzivi.

Vrednost indeksa telesne mase je izračunljiv podatek. Običajno takih podatkov ne shranjujemo, ampak jih izračunamo ob poizvedbi. Kljub temu ga, zaradi lažjega iskanja po podatkih z AQL poizvedbami, ob hkratnem vnosu meritev telesne višine in teže izračunamo in shranimo v EZZ. A izračun ter shranjevanje podatka o ITM nista bila implementirana na vseh mestih v aplikaciji, kjer je možen vnos teh dveh količin, zato je bilo potrebno za stare zapise podatke pred uporabo za kategorizacijo dopolniti. Hkrati se pojavlja problem, da kar naenkrat za kategorizacijo potrebujemo dva vnesena podatka (težo ter višino) in ne le enega. To privede do tega, da lahko za izračun uporabljamo le podatke, ko sta bila za pacienta vnesena podatka teže in višine na isti dan. To pomeni, da je za izračun primernih manj ustreznih podatkov, zato je možno, da so ti manj aktualni. Pojavlja se vprašanje, če bi bilo podatka teže in višine smiselno združevati na določen interval (teden, mesec) in iz njiju izračunati ITM na ravni tega intervala (torej npr.: vnesena teža pred enim tednom in višina na današnji dan; iz teh dveh podatkov izračunamo ITM in mu dodamo časovni žig, ki je povprečje časovnih žigov meritev za izračun).

4.6.2 Pridobivanje podatkov vitalnih znakov iz *Think!Ehr Platform*^{TM}

Prvi del implementacije je obsegal pridobivanje podatke z EHR platforme. Za vsakega od podatkov je bilo potrebno poklicati storitev, ki nam vrača podatke iz EZZ-a. Pridobimo jih z AQL-om (primer predstavljen v zapisu 4.5), ter jih shranimo v listo objektov.

```
1 SELECT c/context/start_time, o/data[at0001]/events[at0002]/data[at0003]/items[at0004]/
      value, o/provider
2 FROM EHR[ehr_id/value='<ehrId>']
3 CONTAINS COMPOSITION c CONTAINS Observation o[openEHR-EHR-OBSERVATION.height.v1]
4 WHERE o/name/value = 'Height/Length' AND EXISTS c/context/start_time
```

Zapis 4.5 Podatki se pridobijo s preprosto AQL poizvedbo. Vrednost "ehrId", ki določa, za katerega pacienta se pridobivajo podatki, se zamenja z ustreznim ID-jem.

Pridobljeni seznam podatkov sestavljajo pari časovnega žiga in meritve. Zaokroženi so na dan natančno in sortirani po času padajoče. Za namen izračuna indeksa telesne mase v primeru, če ta ni bil ustrezno shranjen v EZZ, smo nato podatke shranili v javanski objekt imenovan "PatientVitalsDto", ki vsebuje štiri količinska polja tipa "DvQuantity".

Sprehodili smo se čez vsakega od štirih seznamov vrednosti in v primeru, da datuma še ni bilo v mapi smo ga dodali, ter nato nastavili vrednost ustrezne meritve (zapis 4.6).

```
SortedMap<LocalDate, PatientVitalsDto> vitalsMap = new TreeMap<>();
List<Pair<DateTime, DvQuantity>> heightData = growthDataService.getHeightData(
    patientId);
 for (final Pair<DateTime, DvQuantity> heightPair : heightData)
{
  final LocalDate entryDate = heightPair.getFirst().toLocalDate();
  vitalsMap.computeIfAbsent(entryDate, d -> new PatientVitalsDto());
  vitalsMap.get(entryDate).setHeight(heightPair.getSecond());
}
```

Zapis 4.6 Podatki izmerjeni na isti datum se shranijo v skupni Javanski objekt.

Naslednji korak je priprava podatkov za pošiljanje na odjemalca. Ker le-ta zahteva serije podatkov za vsak tip meritve posebej, je bilo podatke treba ponovno razdružiti in pretvoriti v nove sezname objektov "GrowChartVitalsDataPair" – uporaba "Pair" objekta za serializacijo v Javi ni dobra praksa. Na tem mestu je potrebno izračunati tudi manjkajoče izračune ITM. Postopek je predstavljen v zapisu 4.7.

```
final List<GrowChartVitalsDataPair> bmiData = vitalsDataMap.entrySet().stream()
        .filter(v -> v.getValue().getBmi() != null ||
            (v.getValue().getHeight() != null && v.getValue().getWeight() != null))
        .map(
            v ->
            {
              final PatientVitalsDto dto = v.getValue();
              final DvQuantity bmi;
              if (dto.getBmi() != null)
              {
                bmi = dto.getBmi();
              }
              else
              {
                DecimalFormat df = new DecimalFormat("0.000");
                df.setDecimalFormatSymbols(new DecimalFormatSymbols(Locale.ENGLISH));
                double bmiValue = dto.getWeight().getMagnitude() / Math.pow(dto.
                    getHeight().getMagnitude() / 100, 2);

                bmi = new DvQuantity();
                bmi.setMagnitude(Double.valueOf(df.format(bmiValue)));
                bmi.setUnits("kg/m2");
              }
              return new GrowChartVitalsDataPair(
```

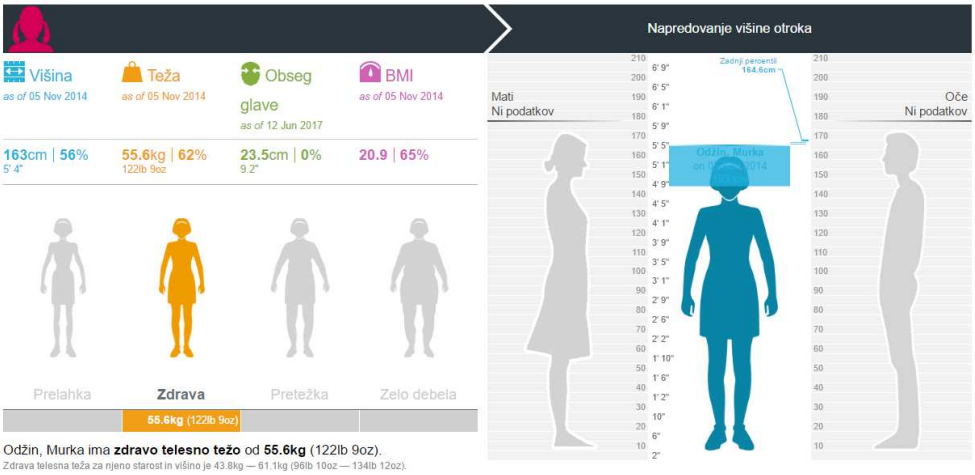
```
24                  formatter.print(v.getKey()),
25                  convertUnits(
26                      bmi.getMagnitude(),
27                      bmi.getUnits(),
28                      "kg/m2"
29                  ));
30              }
31          )
32          .collect(Collectors.toList());
```

Zapis 4.7 Izračun manjkajoče vrednosti ITM. Računamo le v primeru, ko vrednost še ni izračunana. Zaradi izračuna je potrebno izračunano vrednost formatirati, saj drugače pri serializaciji prihaja do težav. Preden podatke pretvorimo v novi objekt, poskrbimo še za morebitno pretvorbo enot. Funkcija za pretvorbo enot "convertUnits" sprejema podatek o vrednosti meritve ter vhodno in izhodno enoto - v tem primeru se pretvorba ne izvede, ker so podatki shranjeni v enakih enotah.

4.6.3 Implementacija kategorizacije z uporabo percentila ITM

Sledila je implementacija same kategorizacije. Aplikacija *Think!Clinical* za prikaz rastnih krivulj uporablja knjižnico "Growth Charts" narejeno s strani podjetja "MedAppTech".

V okviru pričujočega diplomskega dela je bila za namen boljšega izračuna posodobljena funkcionalnost kategorizacije posameznika glede na telesno težo. Prikaz kategorizacije je viden na sliki. 4.7.



Slika 4.7 Prikaz kategorizacije posameznika. V zgornji pasici so prikazane zadnje meritve osnovnih razvojnih vitalnih znakov. V spodnjem delu je izpisan rezultat kategorizacije za posameznika, interval ustrezne telesne teže glede na njegovo višino ter starost in podatek o tem, ali se pacientu stanje izboljšuje ali slabša.

Prvi korak je bila implementacija funkcije, ki izračuna hevristiko z ITM-a. Najprej smo v funkciji pridobili zadnji vnos indeksa telesne mase. Ker so zapisi (podobno kot

v zalednem delu aplikacije) združeni na podlagi datuma, ta vnos vsebuje tudi meritve teže in višine, ki ju potrebujemo za izračun. V primeru, da ITM ne obstaja, vrnemo obvestilo, da ni dovolj podatkov za izračun. Nato izračunamo percentil indeksa telesne mase s funkcijo, ki jo ponuja knjižnica. Ta sprejema poleg osnovnih parametrov vrednosti ITM, starosti in spola tudi zbirko statičnih podatkov za izračun, ki so zapisani v ločeni datoteki. Postopek ponovimo še za predhodno vrednost ITM. Le tega bomo kasneje potrebovali za napoved ali se stanje slabša, ali izboljšuje glede na prejšnjo meritev. S pomočjo knjižnice izračunamo še spodnjo in zgornjo mejo ITM. Minimalni percentil je pet procentov, maksimalni pa petinosemdeset. Iz podatka za minimalni in maksimalni ITM z obrnjeno ITM formulo izračunamo minimalno in maksimalno zdravo telesno težo za prikaz. Nato izvedemo logiko odločanja. Intervali odločanja so sledeči:

- percentil ITM $< 5\%$ - podhranjenost;
- $5\% <$ percentil ITM $> 85\%$ - zdrava telesna teža;
- $85\% <$ percentil ITM $> 95\%$ - debelost;
- percentil ITM $> 95\%$ - huda debelost.

Na vsaki stopnji je prisotna še dodatna logika, ki določa trend, kako se stanje pacienta spreminja s časom. Tu se upoštevata razlika med trenutnim in prejšnjim izračunom ITM ter sama vrednost trenutnega indeksa telesne mase. Vsakega od teh izračunanih podatkov se doda v "javascript" objekt, da lahko funkcija, ki kliče hevristiko, nato ustrezno pokaže izračunane podatke.

V sklopu nadgradnje je bilo potrebno popraviti tudi funkcijo, ki pridobiva podatke za prikaz zgornje pasice podatkov o teži, višini, obsegu glave in ITM, prikazane na sliki 4.7. Potrebno je prikazati meritve višine ter teže, iz katerih je bil izračunan ITM in s katerima je bila kasneje narejena kategorizacija, in ne zadnjih razpoložljivih meritev, kot je bilo narejeno pred nadgradnjo, da ne prihaja do zmede.

5 Zaključek

Informatizacija je za razvoj zdravstva ključnega pomena. Zaradi vse večje količine kliničnih podatkov je potrebno z njimi skrbno ravnati. Z vzpostavitvijo interoperabilne hrbtenice je slovensko zdravstvo naredilo velik korak naprej. Strukturirani, standardizirani podatki zbrani v skupnem repozitoriju, zasnovanem na podlagi mednarodnih standardov IHE, HL7 CDA ter OpenEHR, ki omogočajo souporabo ter moderne klinične raziskave, so že dolgo časa v planu, sedaj pa le-ti plani počasi prehajajo v prakso.

Eno od glavnih podjetij pri vzpostavitvi IH je tudi podjetje Marand d.o.o. V sklopu pričujočega diplomskega dela je bil izdelan pomemben prispevek k njihovi aplikaciji *Think!Clinical*. Izdelane so bile arhetipsko vezane vnosne forme za vnašanje podatkov o oceni stanja, ter nadgrajen modul za oceno stanja debelosti mladostnikov.

Prikazan je bil eden od načinov za agilen razvoj vnosnih form. Ker so splošne predloge in arhetipi že vnaprej izdelane in dostopne v centralnem registru CKM, je možno razviti preprosto vnosno masko brez posebnih funkcionalnosti v roku nekaj ur ali še prej. V prvi fazi je potrebno izdelati predlogo oz. vsaj že narejeni predlogi dodati omejitve ter prevode za ustrezni jezik. Nato z načrtovalcem vnosnih form zgradimo njen opis, ki ga

nato podamo generatorju form, ki nam avtomatsko generira vnosno formo. Potreben je le še zadnji korak in sicer shranjevanje. Vrednosti, ki nam jih ponuja generator, pošljemo na ustrezen "endpoint" in osnovni scenarij je izveden. Tovrstni razvoj je bistveno hitrejši od klasičnega razvoja vnosnih form, saj je pri njem potrebno vsako polje ročno dodati na zaslonsko masko, zanj implementirati ustrezno validacijo, ob koncu pa zbrati vrednosti vseh polj ter jih shraniti, nato pa še reševati možne napake pri strežniški validaciji.

Seveda pa ima vsak uspešen produkt tudi svoje pomanjkljivosti. Generator form trenutno ne podpira prikazovanja vsebine v več stolpcih. Prav tako na primer nima podpore za pogled "accordion"-a. Vsako tovrstno dopolnitev je možno rešiti z dodajanjem kaskadnih slogov in s tem razširiti njegovo delovanje. Prav tako je slišati veliko različnih odzivov na sam vizualni izgled generatorja. Mnogi, predvsem starejši uporabniki, se ne strinjajo, da le-ta z novim oblikovanjem prinaša napredek v razvoju vnosnih mask napram starim vnosnim formam implementiranih v javanskem ogrodju "Swing". Tudi ta problem smo reševali v sklopu te diplomske naloge. Lep primer je popravek na slogih za prikaz zavihkov, ki v osnovi sploh še niso bili definirani, saj je šlo za novo, še nikjer uporabljeno funkcionalnost. Poleg tega se pojavljajo še težave s hitrostjo delovanja tovrstnih form, ki delujejo kvečjemu nekoliko počasneje kot klasične javanske forme. Prostor za izboljšave še vedno obstaja, je pa vseskozi težava v pomanjkanju delovnega kadra ter nenehnem razvoju tehnologij. V demo fazi je namreč že novi generator razvit z ogrodjem "Angular2", ki naj bi odpravil tudi pomanjkljivosti starega generatorja.

Izzivi in vizije ostajajo tudi pri sami implementaciji form v *Think!Clinical* opravljeni v sklopu pričujočega diplomskega dela. Največ vprašanj se poraja predvsem pri formi za vnos ocene stanja pacienta (vitalni znaki - povzeta forma). Glavna težava tukaj je dvostopenjska validacija in potrjevanje form.

Ob prenovi form se namreč ni kaj preveč razmišljalo o nadgradnji uporabniške izkušnje, ampak predvsem o prehodu na nove, modernejše tehnologije. Dvostopenjska validacija namreč precej hitro zbega uporabnika. Nekateri od arhitektov aplikacije so mnenja, da je forma povzetek odveč, ter, da bi bilo potrebno formo razbiti na več podform, ki bi neodvisno od drugih skrbele za shranjevanje podatkov. S tem bi predvsem razbili kompleksnost vnašanja podatkov. Težava je tudi glede samega nabora podatkov na formah. Razširjena forma vitalnih znakov po mnenju nekaterih ne vsebuje le polj za vnos vitalnih znakov pacienta, ampak še mnogo drugih (npr.: oceno izločenih tekočin, urina, blata, bruhanja...), ki bi morali biti tudi ločeni na svoji samostojni vnosni maski.

Spet drugi zagovarjajo formo povzetek ocene stanja, češ da omogoča hiter vnos izmerjenih količin z več različnih področij na skupni formi in tako pospešuje delovni proces vnosa, kar dejansko je njena prednost. Na tej točki se je potrebno zavedati tudi tega, da uporabniki aplikacije niso preveč odporni na večje spremembe v delovanju aplikacije in bi jih prevelike spremembe zmedle pri njihovem delu, ter ga s tem upočasnile. Vsako spremembo je zato potrebno izvesti v manjših postopnih korakih.

Izboljšava na prikazu izračuna kategorizacije mladostnikov je za realne vnesene podatke kar precejšnja, saj zelo dobro kategorizira poleg povprečnih tudi ekstremnejše primere (visoke ter težje oz. nižje ter lažje od povprečja), ki so bili prej obravnavani kot predebeli ali presuhi. V teh primerih so morali pred popravkom zdravniki sami presoditi, zakaj je prišlo do take kategorizacije. Z novo funkcionalnostjo bo takih primerov manj, je pa res, da je končna odločitev še vedno zdravnikova in edino pravilno je tako. Več pa bo pokazala dolgotrajna uporaba v produkcijskem okolju.

Povedano na kratko, agilen razvoj arhetipsko vezanih form z ustreznimi orodji ima veliko prednosti pred klasičnim razvojem. Kot vsako običajno ogrodje tudi generator form zahteva ustrezno integracijo v poljubno (spletno) aplikacijo. Za zahtevnejše komponente in poglede je potrebno smotrno presoditi, ali nalogo izpeljati na tak način, ali po običajnih postopkih. Vsekakor so taka orodja velik doprinos k hitremu razvoju in širitvi (zdravstvenih) informacijskih sistemov, ki se bodo v prihodnosti še nekaj časa hitro razvijali.

Literatura

[1] dr. Dalibor Stanimirović. Informatizacija v zdravstvu. http://www.nijz.si/sites/www.nijz.si/files/uploaded/informatizacija_v_zdravstvu_2.pdf, 2017. [Online; accessed 7-March-2017].

[2] Oddelek za omrežja, vsebino in tehnologije - Evropska komisija. eHealth - Digital Single Market. https://ec.europa.eu/digital-single-market/en/policies/ehealth, 2014. [Online; accessed 11-July-2017].

[3] NIJZ. e-Zdravje. http://www.ezdrav.si/ezdravje/, 2017. [Online; accessed 7-March-2017].

[4] NIJZ. e-Zdravje. http://www.nijz.si/sl/ezdravje, 2017. [Online; accessed 15-May-2017].

[5] Ministrstvo za zdravje RS. Interoperabilna hrbtenica. http://nio.ezdrav.si/?page_id=94, 2017. [Online; accessed 7-March-2017].

[6] Ministrstvo za zdravje RS. Interoperabilna hrbtenica. http://nio.ezdrav.si/?page_id=116, 2017. [Online; accessed 6-April-2017].

[7] NIJZ. Povzetek podatkov o pacientu (PPoP). http://www.nijz.si/sl/povzetek-podatkov-o-pacientu-ppop, 2016. [Online; accessed 5-May-2017].

[8] Ringholm. History of HL7 RIM. http://www.ringholm.com/docs/04500_en_History_of_the_HL7_RIM.htm, 2011. [Online; accessed 7-April-2017].

[9] Organizacija OpenEHR. Origins of OpenEHR. http://www.openehr.org/about/origins, 2017. [Online; accessed 9-April-2017].

[10] Joint Working Group for a Common Data Model. Trial-Use Standard for Healthcare Data Interchange - Information Model Methods. http://www.meb.uni-bonn.de/standards/IEEE/JWG-MODEL/stdmain.pdf, 1996. [Online; accessed 2-July-2017].

[11] George W. Beeler et al. HL7 Reference Information Model. https://www.hl7.org/documentcenter/public_temp_358EBC9C-1C23-BA17-0CF40E625AF7B053/calendarofevents/himss/2011/HL7%20Reference%20Information%20Model.pdf, 2017. [Online; accessed 9-April-2017].

[12] The CEN13606 Association. The CEN/ISO EN13606 standard. http://www.en13606.org/the-ceniso-en13606-standard, 2011. [Online; accessed 7-April-2017].

[13] Wikipedia. Health Level 7. https://en.wikipedia.org/wiki/Health_Level_7, 2017. [Online; accessed 8-April-2017].

[14] Wikipedia. Clinical document architecture. https://en.wikipedia.org/wiki/Clinical_Document_Architecture, 2017. [Online; accessed 8-April-2017].

[15] Borut Fabjan. Using OpenEHR platform and HL7/IHE interoperability. http://www.myphr.com/StartaPHR/what_is_a_phr.aspx, 2014. [Online; accessed 18-June-2017].

[16] Organizacija OpenEHR. OpenEHR block diagram. http://www.openehr.org/releases/BASE/latest/docs/architecture_overview/diagrams/openehr_block_diagram.png, 2017. [Online; accessed 16-April-2017].

[17] Organizacija OpenEHR. What is OpenEHR? http://www.openehr.org/what_is_openehr#, 2017. [Online; accessed 15-April-2017].

[18] Organizacija OpenEHR. What is OpenEHR? http://www.openehr.org/files/what_is_openehr/openehr_dev_ecosystem.png, 2017. [Online; accessed 16-April-2017].

[19] Omowizard. Archetypical. https://omowizard.files.wordpress.com/2010/01/bp-mindmap.jpg, 2010. [Online; accessed 8-July-2017].

[20] Iljaž, Rade. Elektronski zdravstveni zapis (EZZ) = Electronic health record (EHR). *Bilten-ekonomika, organizacija, informatika v zdravstvu*, 25(5):157–158, 2009.

[21] Organizacija OpenEHR. OpenEHR architecture overview. http://www.openehr.org/releases/BASE/latest/docs/architecture_overview/diagrams/security_features.png, 2017. [Online; accessed 16-April-2017].

[22] OpenEHR. The EHR System. http://www.openehr.org/releases/BASE/latest/docs/architecture_overview/architecture_overview.html#_the_ehr_system, 2016. [Online; accessed 23-June-2017].

[23] Peter Garrett in Joshua Seidman. EMR vs EHR. https://www.healthit.gov/buzz-blog/electronic-health-and-medical-records/emr-vs-ehr-difference/, 2011. [Online; accessed 18-June-2017].

[24] AHIMA fundation. What is personal health record? http://www.myphr.com/StartaPHR/what_is_a_phr.aspx, 2017. [Online; accessed 18-June-2017].

[25] Paul C. Tang, MD et al. Personal Health Records: Definitions, Benefits, and Strategies for Overcoming Barriers to Adoption. *JAMIA - The Journal of the American Medical Informatics Association*, 13(2):121–126, Mar-Apr 2006.

[26] Organizacija OpenEHR. Composition structure. http://www.openehr.org/releases/BASE/latest/docs/architecture_overview/diagrams/composition_structure.png, 2017. [Online; accessed 16-April-2017].

[27] Peter Schloeffel, Thomas Beale et al., Ocean Informatics Pty Ltd. The Relationship between CEN 13606, HL7 and OpenEHR. http://www.academia.edu/21215797/The_Relationship_between_CEN_13606_HL7_and_OpenEHR, 2006. [Online; accessed 9-April-2017].

[28] David Cohen, Mikael Lindvall in Patricia Costa. Agile software development. *DACS SOAR Report*, 11, 2003. URL http://users.jyu.fi/~mieijala/kandimateriaali/Agile%20software%20development.pdf.

[29] Organizacija OpenEHR. Overview - the development process. http://www.openehr.org/releases/AM/latest/docs/Overview/Overview.html#_the_development_process, 2014. [Online; accessed 24-June-2017].

[30] Organizacija OpenEHR. Template object model. http://openehr.org/releases/trunk/architecture/am/tom.pdf, 2017. [Online; accessed 3-Maj-2017].